ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería del software

Sistema escalable de planificación horaria para eventos
Scalable schedule planning system for events

Realizado por
SERGIO RAMÍREZ PÉREZ

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACION

UNIVERSIDAD DE MÁLAGA
Málaga, septiembre de 2018

Fecha de defensa:
El Secretario del Tribunal

Resumen

Existen multitud de herramientas que tratan de resolver problemas de planificación horaria de eventos, tales como decidir qué día y a qué hora es mejor su realización. Estas dejan de ser útiles cuando su curva de aprendizaje es demasiado elevada, no son escalables a un gran número de participantes o no son lo suficientemente flexibles para representar cualquier circunstancia. Por este motivo surge la necesidad de desarrollar este proyecto, supliendo la carencia de aplicaciones que realicen esta tarea de una forma sencilla, flexible y escalable.

Con el objetivo anterior, se establecerán unos requisitos muy estrictos de sencillez de uso, flexibilidad y escalabilidad tanto por parte del backend, permitiendo así proporcionar una API que se encargue de toda la lógica necesaria para la planificación horaria del evento mediante una interfaz simple de usar, y frontend, proporcionando al usuario una herramienta amigable con la que interactuar de forma intuitiva para representar las condiciones de su evento, por muy específicas que sean, seleccionar su disponibilidad y ver rápidamente cuál es la mejor fecha para la realización del mismo.

Finalmente, dado que la API será pública y cualquier persona podrá usarla, permitirá la centralización de cualquier proyecto para la planificación de eventos en un mismo servicio, de forma que podamos interactuar con un evento desde una plataforma distinta a la que se usó para crearlo. Por ejemplo, nos permitiría desarrollar un bot para la aplicación “Telegram” que permita a los participantes seleccionar rápidamente la fecha más conveniente, pero si desean realizar una selección más precisa, pueden hacerlo abriendo el mismo evento desde una aplicación Android o un navegador web.

Palabras claves

Planificación horaria de eventos, React, Node.js, MongoDB

Abstract

There are many tools that try to solve events schedule planning problems, such as deciding which day and at what time it is better to perform it. These are however no longer useful when their learning curve is too high, are not scalable to many participants or are not flexible enough to represent any circumstance. For this reason, the need to develop this project arises, supplying the lack of applications that perform this task in a simple, flexible and scalable way.

Accordingly, very strict requirements of simplicity of use, flexibility and scalability will be established both in the backend, thus allowing to provide an API that takes care of all the logic necessary for the schedule planning of the event through a simple to use interface, and frontend, providing the user with a friendly tool to intuitively interact and represent the conditions of his event, no matter how specific they are, select his availability and quickly see which is the best date to carry it out.

Finally, since the API will be public and anyone can use it, it will allow the centralization of any events schedule planning project in the same service, allowing the user to interact with an event from a platform different from the one used to create it. For example, it would allow us to develop a bot for the application “Telegram” that allows participants to quickly select the most convenient date, but if they wish to make a more precise selection, they can do it by opening the same event from an Android application or a web browser.

Keywords

Event schedule planning, React, Node.js, MongoDB

Índice

Índice de figuras

1. Introducción

1.1. Motivación y objetivos

Para realizar cualquier evento, ya sea una quedada informal, una reunión de negocios o un gran evento internacional, es necesario ponerse de acuerdo con los participantes para decidir qué fecha y hora es mejor la realización del mismo. Históricamente esta tarea se ha llevado a cabo hablando con todos los participantes, sin embargo, la necesidad de crear este proyecto surge cuando disponen de poco tiempo libre y es complicado elegir una fecha donde todo el mundo pueda asistir, o cuando el número de participantes es lo suficientemente elevado como para que no se pueda gestionar fácilmente preguntando a cada uno de ellos.

Por este motivo se necesita una herramienta que ayude a decidir la mejor fecha para realizar el evento, sin embargo, para ser realmente útil en todas las condiciones, consideramos prioritario que esta herramienta disponga de tres requisitos fundamentales:

- **Sencillez.** Dado que se trata de una herramienta cuyo propósito principal es facilitar la tarea de realizar un evento, la curva de aprendizaje debe ser mínima.
- **Flexibilidad**. Debe poder adaptarse a cualquier situación, por muy específica que sea, pudiendo así ser utilizado en cualquier tipo de evento.
- **Escalabilidad.** Un evento puede tener desde tan sólo un par, hasta miles de participantes, por lo que es necesario que la herramienta pueda ser escalarse sin sacrificar ninguno de los requisitos anteriores.

Hoy día disponemos de herramientas que intentan solventar el problema, pero tras estudiar las soluciones existentes, no hemos encontrado ninguna que cumpla con los tres requisitos fundamentales expuestos anteriormente. Por este motivo, el objetivo de este proyecto es desarrollar una solución definitiva a la planificación horaria de eventos, siendo prioritario el cumplimiento de los requisitos en cada componente del sistema.

1.2. Estudio de las soluciones existentes

A continuación, vamos a analizar las mejores soluciones y representativas de las distintas alternativas existentes a la fecha de publicación de este documento en base a los requisitos fundamentales planteados anteriormente.

1.2.1. Doodle

Es la herramienta más utilizada con más de 10 millones de usuarios. Se basa en un sistema de opciones, donde un usuario crea un evento seleccionando las distintas opciones que considera adecuadas y los participantes eligen aquellas que mejor se cuadra con su disponibilidad. Esto lo hace de una forma muy rápida y sencilla, sin necesidad de un registro previo, además de permitir a un número ilimitado de participantes que seleccionen la opción que deseen, esto hace que cumpla dos de los tres requisitos. Sin embargo, no cumple el requisito de flexibilidad, ya que tan sólo permite seleccionar el horario a la persona que crea el evento, siendo imposible para los participantes seleccionar un horario más específico (dentro de las opciones permitidas).

Por ejemplo, si la persona que crea el evento selecciona 2 opciones por cada día, una por la mañana y otra por la tarde, el participante no podrá seleccionar qué hora de la mañana o tarde tiene disponible. Si por el contrario, en la creación del evento se selecciona tramos de una hora, además de tener el problema anterior, donde el participante sólo puede elegir un tramo de una hora como mínimo, nos encontramos con que deja de ser fácil de usar, incumpliendo con uno de los requisitos fundamentales expuestos en este documento, ya que tendríamos 24 opciones a elegir por cada uno de los días, de forma que no sólo compila la selección de la disponibilidad de los participantes, sino que deja ser fácil ver cuál es la mejor más recomendada.

A continuación, podemos ver en la figura 1 un evento creado en Doodle representando el ejemplo, permitiendo a los participantes una flexibilidad mayor, aunque quizá no suficiente para alguno, donde se pueda seleccionar tramos de una hora, durante dos días. Como puede observarse a simple vista, deja se ser intuitivo y fácil de usar para ser un poco más caótico.

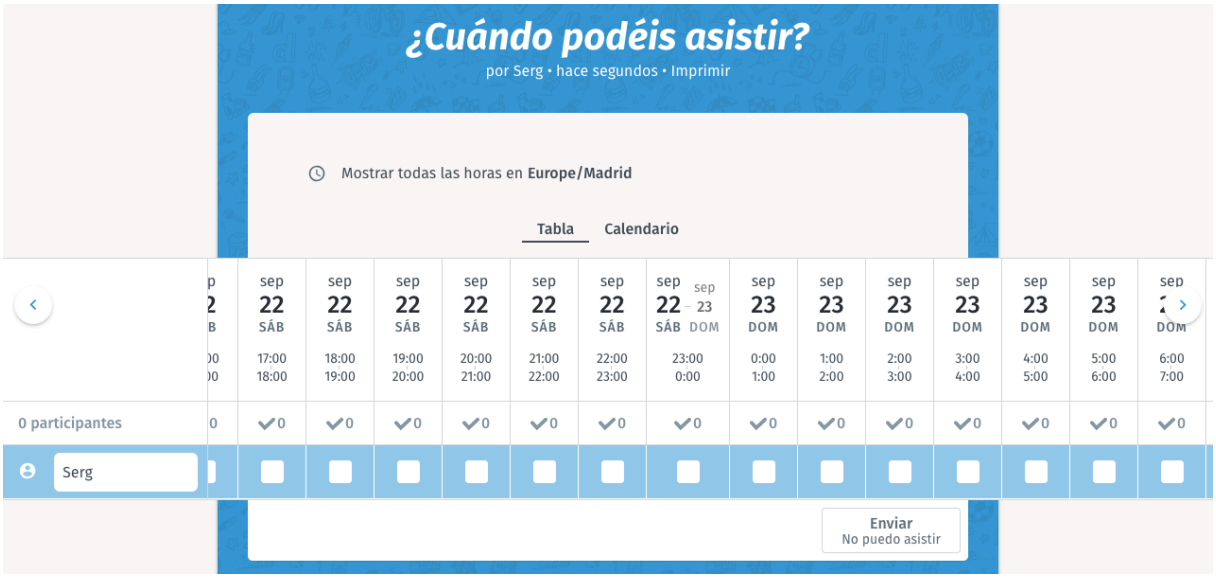


Figura 1. Doodle (https://doodle.com)

1.2.2. Vyte

Es la solución más completa que hemos encontrado tras Doodle. Al igual que el anterior, nos permite crear eventos mediante un sistema de opciones, aunque en esta ocasión nos permite una serie de característica que Doodle no nos permite, como un sistema de mensajería incorporado en cada evento o dejar que sean los participantes los que nos propongan las fechas. En cuanto a la facilidad de uso, no es está al nivel de otras soluciones como Doodle, debido a que requiere un registro previo (únicamente mediante OAuth2) y el evento no se crea desde la misma página inicial, sin embargo, una vez que nos registramos y comenzamos a crear el evento, el proceso es muy sencillo, lo mismo ocurre a la hora de participar, tan sólo hay que selecciona la opción más adecuada y ya hemos terminado.

En cuanto a escalabilidad y flexibilidad, dado que usa el mismo sistema de opciones que hemos comentado anteriormente, podemos decir que es ampliamente escalable y muy poco flexible, permitiendo únicamente seleccionar las opciones que se han escogido anteriormente o sacrificando la facilidad de uso a cambio de permitir una mayor flexibilidad con un mayor número de opciones cada día.

Para ejemplificar el problema nuevamente, se ha creado un evento con las mismas condiciones que en el caso anterior, permitiendo una opción por cada hora del día durante dos días, obteniendo como resultado el evento que se muestra en la figura 2.

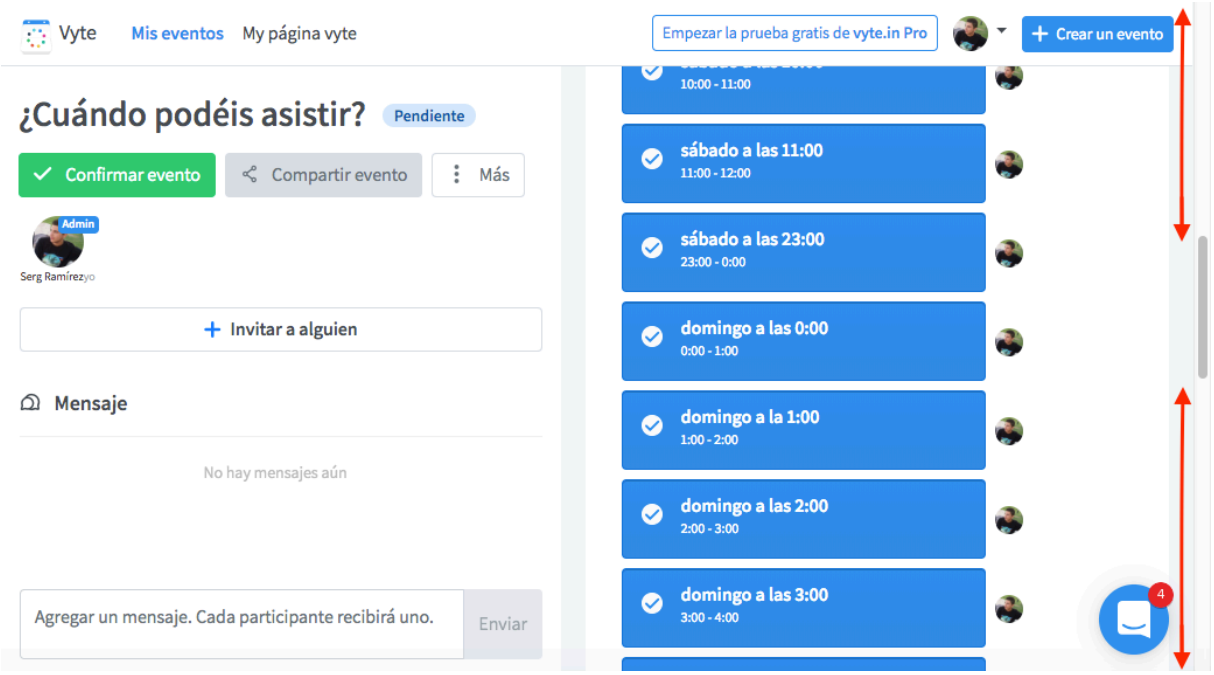


Figura 2. Vyte (https://www.vyte.in)

Como punto positivo, esta solución es la única entre todas las aplicaciones que analizaremos que dispone de una API abierta a terceros.

1.2.3. Meetomatic

Se trata de una aplicación más simple a las soluciones analizadas anteriormente, pero con la posibilidad de seleccionar fechas anteriores a la actual, además de no necesitar registro previo. No obstante, es necesario introducir un correo electrónico tanto para crear como para participar en el evento, pero dado que tan sólo se necesita un paso, su punto fuerte es la facilidad de uso.

No obstante, dispone de muy poca flexibilidad y escalabilidad, ya que, además de basarse en el sistema de opciones anteriormente analizado, la opción de seleccionar tramos horarios está reservada exclusivamente a usuarios de pago (muy poca flexibilidad) y en lugar de mostrarnos los datos por cada opción, donde se vea claramente qué día tiene una mayor disponibilidad, se muestra por participantes, de forma que al participar un gran número de usuarios, sería muy complicado ver de una forma rápida cuáles serían las mejores opciones (muy poca escalabilidad).

A continuación, podemos ver en la figura 3 la forma de representar los datos y donde puede verse el problema que supondría si participase un gran número de usuarios.

VIP?		= resend personal meeting link	Mon 3/Sep	Tue 4/Sep	Wed 5/Sep	Thu 6/Sep	Fri 7/Sep	
☐	Abernathy	abernathy.magnus867@mailbox87.de	✖	✔	✖	✔	✖	
☐	Prohaska	prohaska.kaelyn957@mailbox87.de	✖	✔	✔	✔	✖	
☐	Dejon	dejon.moore890@mailbox87.de	✔	✖	✔	✖	✖	
☐	Jeremy	jeremy.treutel334@mailbox87.de	✖	✔	✖	✖	✔	
	Rank			1				

Figura 3. Meetomatic (http://www.meetomatic.com)

1.2.4. WhenIsGood

Se trata de un servicio de planificación de eventos con un sistema distinto a los usados por las soluciones analizadas hasta ahora, ya que muestra las opciones en un calendario donde se pueda ver qué horas son las más seleccionadas por los participantes. Este sistema es el más conveniente si se desea una mayor flexibilidad, sin embargo, consideramos que no ha sido correctamente implementado, ya que además de tan sólo permitirnos un tramo mínimo de 15 minutos (quizá insuficiente para algunos usuarios); la forma de representarse es demasiado detallada, haciendo que sea difícil ver a simple vista cuáles son las horas más seleccionadas. Por este motivo la facilidad de uso se ve claramente afectada.

Además, nos muestra el número de participantes que han seleccionado cada opción con un sistema de puntos (ver figura 4) que hace que sea muy difícil ver claramente cuáles son las opciones más solicitadas en caso de tener un gran número de participantes, lo que afecta a la escalabilidad.

No obstante, permite una mayor flexibilidad que las soluciones anteriores, ya que se puede representar de una forma sencilla una serie de tramos horarios en los que se podría llevar a cabo el evento y cada participante puede seleccionar cualquier otro tramo dentro de los disponibles.

A continuación, podemos ver en la figura 4 un evento donde se ha seleccionado unos tramos horarios más específicos (de 10:00 a 14:45 y de 17:00 a 17:30) para cada uno de los 16 días disponibles y donde un usuario que desee participar pueda seleccionar aquellos tramos que mayor se corresponda con su disponibilidad.

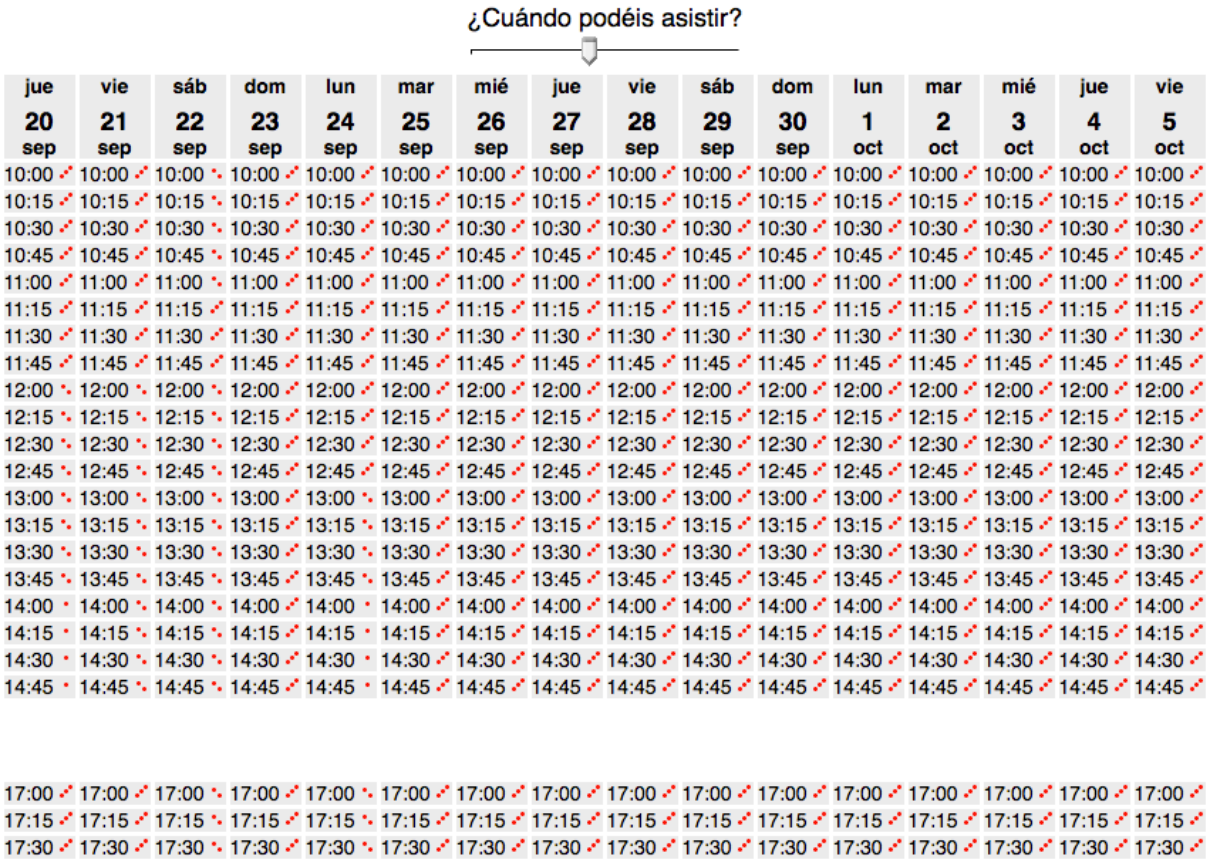


Figura 4. WhenIsGood (http://whenisgood.net)

1.3. Tecnologías utilizadas

La aplicación se ha desarrollado íntegramente en JavaScript, un lenguaje imperativo basado en prototipos, débilmente tipado e interpretado con una gran acogida debido a su facilidad de uso, versatilidad y velocidad gracias al motor V8 Engine principalmente, motor en el que se basa el navegador Chrome y el entorno Node.js.

En el gráfico de la figura 5 se puede ver una evolución del uso de los lenguajes más comunes y cómo JavaScript tiene un mayor porcentaje de uso que otros como Java, PHP o C#.

Una de las características de JavaScript que le proporcionan esa versatilidad es no tener una única sintaxis, sino que existen distintas especificaciones, siendo ECMAScript la más utilizada, aunque podemos usar otras como TypeScript o CoffeeScript entre otras. En este proyecto se ha usado dos especificaciones distintas, para el backend se ha usado el entorno Node.js con el estándar ECMAScript y React

con la especificación JSX (una extensión de ECMAScript con elementos cuya sintaxis es similar a HTML) para el frontend.

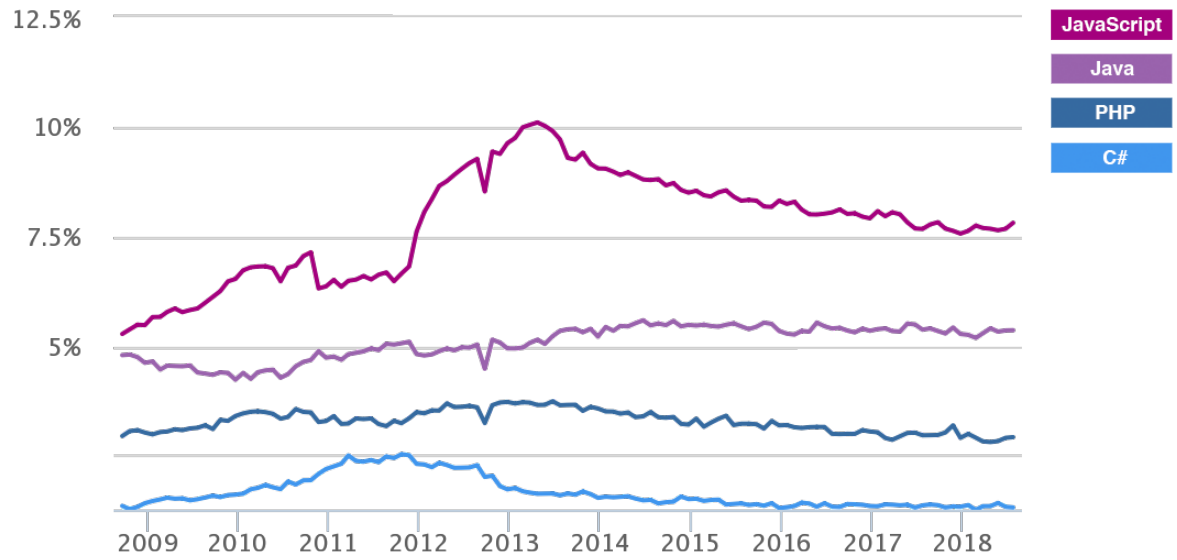


Figura 5. Uso de lenguajes (https://www.openhub.net/languages/compare)

1.3.1. Node.js

Se trata de un entorno basado en el motor de JavaScript V8 que ha permitido la creación de aplicaciones de red altamente escalables usando este lenguaje fuera del entorno de los navegadores web, proporcionando una gran cantidad de librerías para permitir interactuar directamente con el sistema operativo y el hardware de la máquina en la que se ejecuta.

Además, cuenta con una gran comunidad que a la fecha de publicación de este documento ha publicado más de 700.000 paquetes en NPM, el gestor de paquetes por defecto de Node.js y el más grande existente hasta la fecha.

Por este motivo y, dado que se desea implementar un servicio REST en la red, se ha seleccionado Node.js como entorno de desarrollo para el backend de la aplicación, beneficiándonos de las ventajas anteriormente comentadas que nos proporciona Node.js y JavaScript.

Con este entorno, se podría crear un servicio REST directamente, aunque existen numerosos frameworks que simplifican en gran medida esta tarea, por lo que, tras realizar un estudio entre las distintas alternativas, finalmente se ha optado por usar

LoopBack, un framework basado en Express.js, otro framework para Node.js muy popular y que nos permite registrar servicios REST de una forma muy sencilla. LoopBack nos permite definir la estructura de los modelos que componen nuestra aplicación, la relaciones entre ellos y los derechos de acceso mediante un JSON para generar automáticamente los servicios REST necesarios para realizar un CRUD (acrónimo de *Create, Read, Update y Delete*) que podemos completar con funciones propias definidas en un archivo JavaScript. Además, nos ofrece una interfaz gráfica con un explorador muy completo de la API donde podemos realizar consultas, gestiona los tokens de inicio de sesión, el acceso a cada servicio de nuestra API, entre muchas otras ventajas.

LoopBack tiene una licencia MIT y la mantiene una gran comunidad, contando con más de 40.000 estrellas en GitHub y más de 7.000 forks a la fecha de publicación de este documento.

1.3.2. MongoDB

Como sistema gestor de la base de datos se ha seleccionado MongoDB, una base de datos no relacional, o NoSQL, basada en BSON, un formato cuyo nombre proviene de Binary JSON y se trata de una representación binaria del formato JSON. El almacenamiento de datos en documentos de MongoDB sacrifica la validación y consistencia de los mismos, pero nos proporciona una mayor rapidez y agilidad, lo que hace que sea ideal para gestionar grandes cantidades de datos que no contengan muchas relaciones entre ellos.

Además, dado que la forma nativa de realizar consultas y obtener los datos de JavaScript, se integra muy fácilmente con la plataforma que hemos seleccionado para el backend de la aplicación, Node.js.

1.3.3. React

Para realizar el frontend, se ha optado por usar React, una de las mayores bibliotecas de código libre y con licencia MIT para el desarrollo de aplicaciones frontend, mantenida por Facebook y una gran comunidad, contando con más de 111.000 estrellas en GitHub y casi 20.000 forks a la fecha de publicación de este documento.

React nos permite desarrollar una aplicación web que se ejecuta en el navegador del cliente, liberando de una gran carga al servidor, el cual únicamente debe encargarse de enviarlo (tan sólo una vez, ya que se almacenará en la caché del navegador) y responder a las peticiones REST del mismo.

Esta biblioteca se basa en componentes web, una forma de estructurar el contenido de la aplicación y encapsularlo, de forma que se facilita la reutilización de código y permite integrar de una forma muy sencilla componentes de terceros. En nuestro caso, se ha optado por seguir las normas de diseño definidas por las directivas de Material Design, unas normas definidas por Google para unificar el diseño del sistema operativo Android, y para ello se ha utilizado un conjunto de componentes llamado Material-UI, mantenido por un pequeño equipo y una gran comunidad, cuyo repositorio se puede encontrar en https://github.com/mui-org/material-ui

A partir de esta biblioteca, componentes propios y de terceros, se ha desarrollado una interfaz que cumpla con los requisitos fundamentales que se especificó en el capítulo 1.1. de este documento.

1.4. Metodología de desarrollo

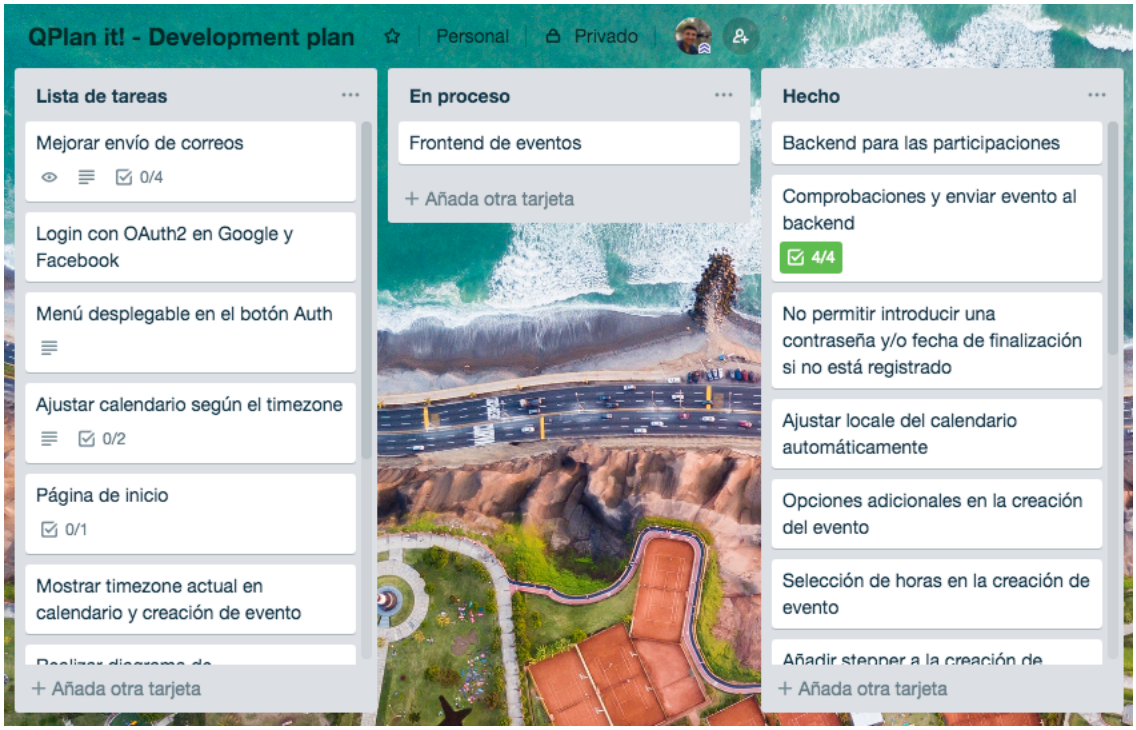


Figura 6. Tablero de Trello (http://trello.com)

El proceso de desarrollo que se ha seguido ha sido iterativo incremental, manteniendo en todo momento una base en funcionamiento, esto se ve reflejado en los commits del repositorio Git del proyecto. Debido a la naturaleza de las metodologías ágiles, el proyecto ha ido evolucionando constantemente y surgiendo cambios relevantes, por lo que, para realizar una descripción más precisa, se comentará las iteraciones más relevantes basándose en el resultado final.

Además, para cada iteración se definió una serie de tareas que se han ido situando en tres listas según corresponda, “To do”, “Doing” y “Done”, usando la herramienta Trello, tal y como puede observarse en la figura 6.

1.4.1. Primera iteración: Análisis de herramientas disponibles

En primer lugar, se realizó un análisis de las herramientas disponibles que proporcionen una solución al problema planteado, detectando sus carencias y anotando los puntos fuertes a su vez, esto nos proporcionó una visión más clara de cómo debe ser el producto final que se desea conseguir. Esta iteración duró aproximadamente 5 horas.

1.4.2. Segunda iteración: Diseño de prototipos

Una vez que se tenía una visión de lo que se desea desarrollar, se realizó una nueva iteración en la que se diseñaron unos prototipos interactivos, de forma que no sólo ayude a especificar qué se desea desarrollar de una forma mucho más clara, sino que nos permita detectar posibles carencias o fallos de concepto antes de realizar ninguna implementación. Estos prototipos, o mockups, se realizaron con la herramienta Balsamiq Mockups, y dado que se requería un concepto que cumpliese los requisitos que especificamos al inicio de este documento, pero no existía en ninguna de las herramientas estudiadas en la primera iteración, nos ocupó más tiempo del que inicialmente se planteó, durando aproximadamente unas 20 horas.

En la figura 7 se muestra algunos de los prototipos más relevantes, donde se puede observar la página principal, la selección de días del evento que se desea realizar, la configuración horaria y un evento donde ha participado tres personas, observándose fácilmente los mejores tamos horarios para la realización del mismo.

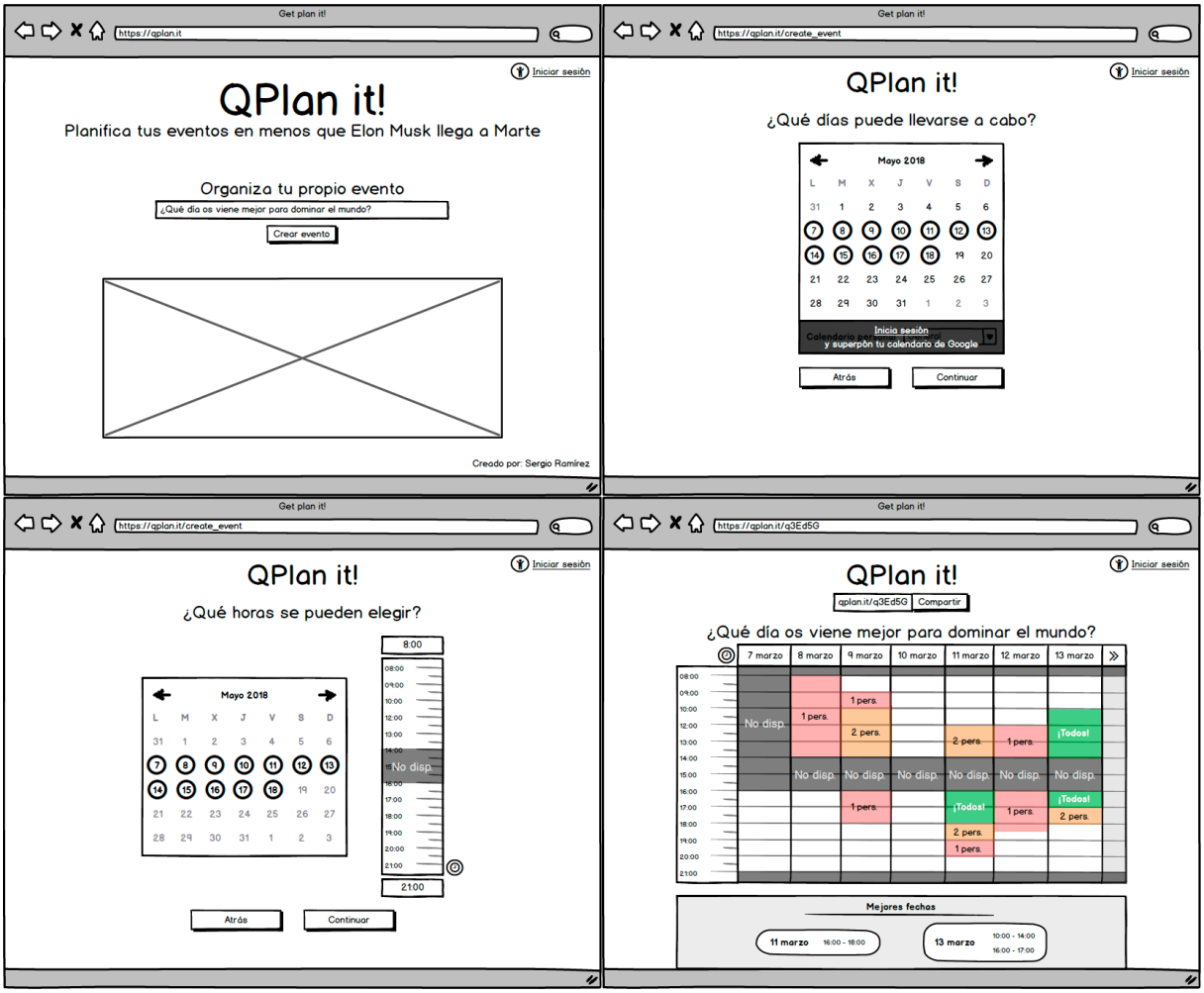


Figura 7. Mockups de la aplicación

1.4.3. Tercera iteración: Modelado UML

Tras definir cómo sería la aplicación en última instancia, se inició una nueva iteración donde se definió cómo sería el backend de la aplicación, por lo que para ello se diseñó un diagrama UML en el que se decidió cómo se estructuraría la aplicación, qué atributos y operaciones contendría cada uno de los módulos y cómo se relacionan e interactúan entre ellos. Además, se modeló la estructura mediante un diagrama de entidad-relación (ERD) partiendo del diagrama de clases previamente modelado (más adelante se comentará con más detalles la estructura del backend y los diagramas).

Tal y como se comentó al inicio de este apartado, debido a la naturaleza de las metodologías ágiles, se ha ido realizando ciertas modificaciones al modelo UML de la aplicación en iteraciones posteriores, por lo que es no es fácil especificar una duración precisa de esta iteración, pero se estima una duración aproximada de 25 horas.

1.4.4. Cuarta iteración: Implementación del backend

Dado que en este punto ya se tiene un modelo UML con la estructura del backend y unos prototipos representando el frontend, se puede comenzar con la implementación. Lo primero que se hizo fue definir cómo sería la interfaz de la API REST, la ruta de cada uno de los servicios, la estructura de los datos que reciben y la estructura de las respuestas.

Una vez especificado todo lo anterior, comenzó la implementación usando el framework LoopBack, cuyo funcionamiento se desconocía, por lo que se dedicó un tiempo en leer la documentación y comprender cómo implementar lo que necesitamos. Al igual que la iteración anterior y debido a la naturaleza de la metodología que se está usando, el backend sufrió algunas modificaciones en la siguiente iteración, con todo, se calcula una duración aproximada de 90 horas.

1.4.5. Quinta iteración: Implementación del frontend

Esta iteración se comenzó cuando se disponía de una funcionalidad básica en el backend de la aplicación, sin embargo, antes de comenzar con el desarrollo se realizó un estudio sobre qué tecnologías, herramientas, patrones de diseño, incluso estructura del proyecto, era más conveniente utilizar. Una vez decidido se comenzó la implementación, que comentaremos detalladamente en futuros capítulos.

Dado que una de las características principales de este proyecto y diferenciadoras con respecto a soluciones ya existente, es la forma que tiene el usuario de interactuar con el sistema, se ha dedicado más tiempo del que se estimó en primera instancia, teniendo una duración aproximada de 120 horas.

1.4.6. Sexta iteración: Redacción de la memoria

Por último, se ha realizado este documento explicando detalladamente cada fase del proceso de desarrollo del proyecto, para ello, mientras se desarrollaba la aplicación se tomó notas indicando qué sería relevante describir y se anotó todas las fuentes consultadas. Esta iteración ha durado 36 horas, completando las 296 horas requeridas para un proyecto de estas características.

1.5. Estructura del documento

Este documento contiene 5 capítulos. El primero es una Introducción al problema y una breve descripción de sistemas similares al descrito en esta memoria.

En el segundo capítulo se detallan los perfiles de usuario disponibles en la aplicación, además de una serie de requisitos funcionales y no funcionales que deberán estar presente a lo largo del desarrollo del proyecto.

En el tercer capítulo se describen los detalles de diseño e implementación del backend, dividiéndose en 5 apartados para comentar detalladamente la estructura del servidor y la base de datos, la interfaz y funcionamiento de la API REST, la estructura del código fuente y una descripción de las características más destacadas de la implementación.

En el cuarto capítulo se describe los detalles de diseño e implementación del frontend, dividiéndose en 5 apartados para comentar detalladamente el patrón de diseño utilizado, la detección automática de idioma, la estructura del código fuente, el enrutador de la aplicación y una descripción de las características más destacadas de la implementación.

Finalmente, el último capítulo describe las conclusiones finales; tras este se especifican las referencias bibliográficas.

2. Requisitos de la aplicación

Antes de comenzar a diseñar e implementar la aplicación, es importante definir qué perfiles de usuario dispondrá el sistema, además de una serie de requisitos funcionales y no funcionales que deben cumplirse en todo el proceso.

2.1. Perfiles de usuario

Usuario anónimo

Este usuario no necesitará registrarse en la aplicación y podrá crear un evento con una serie de limitaciones, ya que algunas características están reservadas a aquellos usuarios con perfil de usuario registrado. Además, se permite interactuar con cualquier evento, añadiendo una participación en la que podrá añadir y eliminar tantos tramos horarios como desee.

Usuario registrado

Se trata de un usuario que necesita un registro previo, indicando unos datos mínimos tales como el nombre, email y contraseña de acceso. Éste permite realizar todas las tareas que se permite hacer con el perfil de usuario anónimo, con la diferencia de poder usar características exclusivas para este perfil, tales como asignar una contraseña de participación o establecer una fecha límite para participar. Además, podrá modificar un evento creado por él en cualquier momento y seleccionar una fecha definitiva.

2.2. Requisitos funcionales

A continuación, se detallarán los requisitos funcionales de la aplicación, estos son una serie de actividades, comportamientos y/o funciones que debe realizar la aplicación para cumplir con su cometido. Se ha detectado 4 requisitos funcionales principales que engloban otro conjunto de requisitos, formando un total de 17 requisitos y que se detallan a continuación:

1. **Gestionar el usuario.**
    1.1. Se permite registrarse usando un email.
    1.2. Se puede iniciar y cerrar sesión.
    1.3. Se puede seleccionar un nombre para el perfil de usuario anónimo.
2. **Crear un evento.**
    2.1. Todos los perfiles de usuarios pueden crear eventos.
    2.2. Se debe introducir un título.
    2.3. Se puede elegir los días disponibles (pasados y presente y futuros).
    2.4. Se puede seleccionar cualquier conjunto de tramos horarios para cada día de forma independiente.
3. **Gestionar un evento.**
    3.1. Se puede bloquear la participación con una contraseña.
    3.2. Se puede establecer fecha límite de participación.
    3.3. Se puede seleccionar una fecha definitiva para la realización del evento.
4. **Seleccionar disponibilidad en un evento.**
    4.1. Se permite seleccionar sólo los días disponibles.
    4.2. Se permite seleccionar sólo los tramos disponibles para cada día.
    4.3. Un usuario sólo puede modificar su selección de disponibilidad.

2.3. Requisitos no funcionales

Por último, se detallarán los requisitos no funcionales de la aplicación, que se trata de un conjunto de propiedades y/o cualidades que imponen unas restricciones en el diseño e implementación. Se ha definido 7 requisitos no funcionales, que se indican y detallan a continuación:

1. **Facilidad de uso:** La interfaz de la aplicación debe ser intuitiva y permitir crear un evento en un máximo de tres pasos.
2. **Requisitos de velocidad:** La aplicación web debe ser rápida y mostrar un proceso de carga si se necesita esperar una respuesta por parte del servidor.
3. **Informar de lo ocurrido:** Si ocurre cualquier fallo, se debe mostrar la información de lo ocurrido, sin por el contrario se ha finalizado un proceso de forma satisfactoria, se debe notificar al usuario.

4. **Requisitos de disponibilidad:** La aplicación web debe permanecer en caché y mostrarse a pesar de no tener conexión con el servidor.
5. **Persistencia de información:** Si un usuario ha indicado sesión, se debe recordar sus credenciales para la próxima visita. Si, por el contrario, un usuario anónimo ha introducido un nombre para participar, se debe recordarse para evitar introducirlo de nuevo en una próxima participación.
6. **Visualización rápida de la mejor opción:** Se debe poder visualizar fácilmente los mejores tramos horarios para la realización del evento.
7. **Requisitos de escalabilidad:** Debe poder participar un número indefinido de participantes en un evento sin comprometer la facilidad de uso.

3. Backend de la aplicación

El backend de la aplicación es el encargado de toda la lógica del proyecto, proporcionando una API REST que permita a otras aplicaciones actuar como cliente, de forma que compartan un único backend y facilitando la interconexión entre ellas, sin importar que sea una aplicación web, de escritorio, móvil o incluso un plugin para otra aplicación.

En este capítulo se detallarán todos los detalles relativos al diseño, estructura e implementación.

3.1. Estructura del servidor

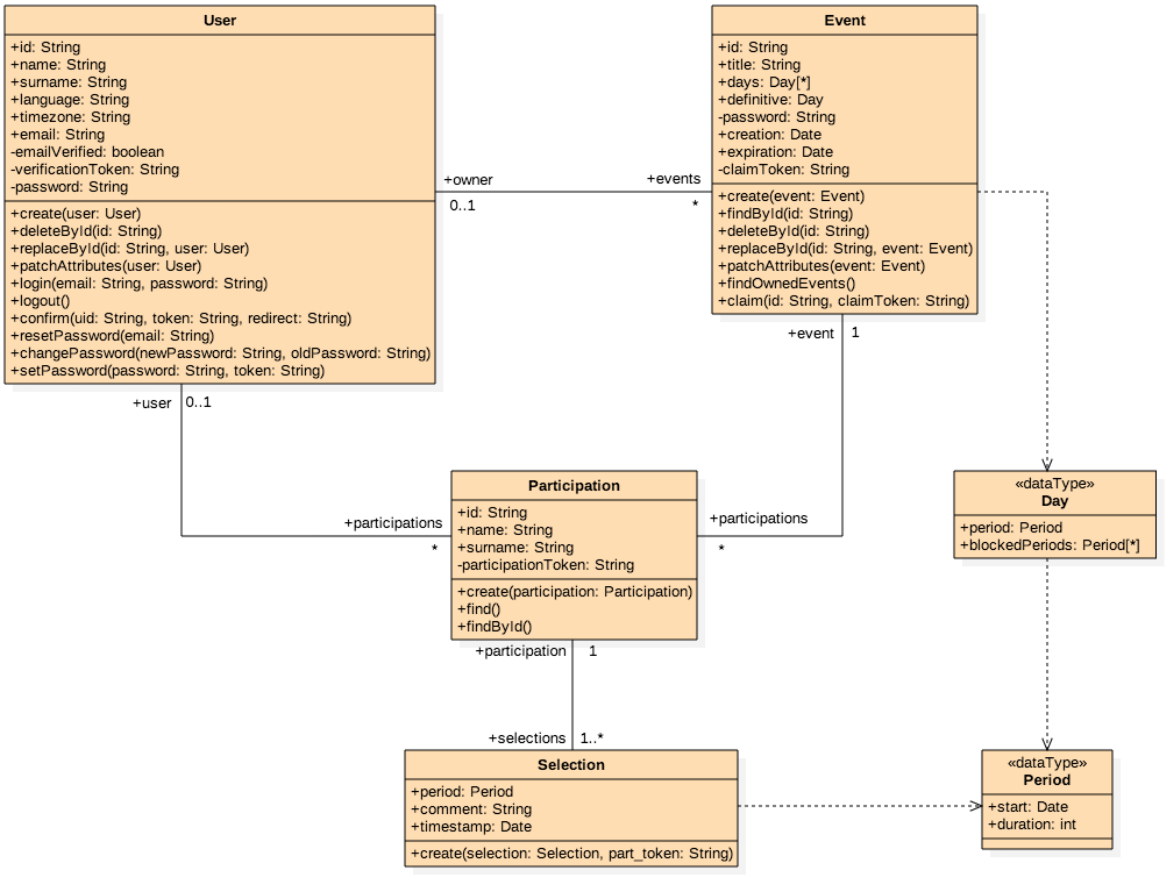


Figura 8. Diagrama de clases UML

Antes de comenzar con la implementación es muy importante definir cómo se estructurará, qué atributos y operaciones contendrá cada objeto o clase y cómo se relaciona con las demás, para ello se ha construido un diagrama de clases usando la herramienta StarUML donde se puede observar claramente la estructura. En la figura 8 se muestra el diagrama comentado y procederemos a explicar los detalles del mismo.

Con el propósito de explicar la estructura modelo de una forma detallada pero concisa, se especificará únicamente las características más importantes, obviando operaciones como el CRUD o atributos básicos como “nombre”, “email” y similares.

Antes de comentar cada una de las clases, nos gustaría indicar que tanto el modelo como todo el código fuente se encuentra íntegramente en inglés, facilitando así la lectura y proporcionando una mayor sensación de unidad en el código.

3.1.1. Clase “User”

Se encarga de mantener la información y proporcionar las operaciones necesarias relativas al usuario, de forma que encontramos una serie de atributos visibles tales como el nombre, apellidos, idioma… de entre los cuales destacamos el atributo “timezone”, que nos permite almacenar una zona horaria para un usuario, de forma que todos los eventos que cree se basen en la misma zona horaria, aunque por defecto se usará la zona horaria del navegador que esté usando. Además encontramos una serie de atributos con visibilidad privada, tales como “password”, que almacena la contraseña cifrada del usuario, “emailVerified”, que nos indica si el usuario ha verificado su dirección de correo electrónico, y “verificationToken”, que consiste en una cadena de texto aleatoria que se genera en el momento de la creación del usuario y se envía por correo electrónico, siendo imposible su posterior lectura, de esta forma podemos intercambiar ese token por una modificación de “emailVerified” a true. Estos atributos son accesibles únicamente desde llamadas en el backend, pero no se podrán acceder desde la API REST.

También dispone de dos relaciones, hacia “Event” y “Participation”, ambas con una

multiplicidad de cero a infinito, lo que indica que un usuario puede tener cero, uno o infinitos eventos o participaciones respectivamente.

Por último, dispone de una serie de operaciones relativas al usuario, de entre las cuales destacamos:

- `login()` recibe un email y contraseña como atributos y comprueba que las credenciales sean válidas, generando un token de acceso, que consiste en una cadena de caracteres única que, al adjuntarla en una petición a la API, proporciona acceso a ciertas operaciones que sería imposible usar sin el token adecuado.
- `logout()` se debe llamar adjuntando un token de acceso creado por la función `login()` y permite eliminarlo, no permitiendo hacer más consultas con ese token y por lo tanto cerrando la sesión.
- `confirm()` nos permite confirmar una dirección de correo electrónico al pasarle como atributos un ID de usuario y un token de validación que se genera en el momento de la creación del usuario y se envía por correo. Además, nos permite introducir una URL de retorno en el caso de que la operación sea satisfactoria.
- `resetPassword()` recibe como atributo un email y, tras comprobar que se encuentra registrado en el sistema, genera un token de acceso temporal (con una caducidad de 15 minutos) y envía un mensaje de correo electrónico a la dirección especificada, adjuntando un enlace a una página que permita cambiar la contraseña usando ese token.
- `changePassword()` recibe como atributos la contraseña actual y una nueva para, tras comprobar que la contraseña actual del usuario coincide, cifrar la nueva contraseña y cambiarla por esta.
- `setPassword()` permite cambiar la contraseña sin necesidad de introducir la actual, para ello necesita un token de acceso que puede ser el que se genera al iniciar sesión o el token temporal que se general al solicitar una nueva contraseña mediante la función `resetPassword()`.

3.1.2. Clase “Event”

Se encarga de mantener la información y proporcionar las operaciones necesarias relativas a un evento, de forma que encontramos una serie de atributos visibles tales

como el ID, título, fecha de creación… de entre los cuales destacamos “days”, que se trata de un array de “Day”, un tipo de dato personalizado que detallaremos más adelante, y que almacena la información de los tramos horarios que se permite seleccionar en el evento, “definitive”, que almacena un tipo de datos “Day” indicando el tramo horario definitivo en el que se realizará el evento, y “expiration”, una fecha a partir de la cual no se permitirá nuevas participaciones. Además, encontramos una serie de atributos con visibilidad privada, tales como “password”, un atributo opcional en el que se almacena una contraseña cifrada para bloquear la participación a cualquier usuario y que por lo tanto se solicitará a la hora de participar, y “claimToken”, una cadena de caracteres aleatoria que se genera y se proporciona cuando un usuario sin registrar crea un evento. Esto permite que el navegador almacene el token y permita usarse para reclamar la propiedad del evento en el caso de que posteriormente decida registrarse o iniciar sesión.

También dispone de dos relaciones, la primera de ellas, llamada “owner”, es hacia la clase “User” e indica quién es el propietario del evento, sin embargo, esta relación tiene una multiplicidad de cero a uno, permitiendo que se pueda crear un evento sin necesidad de un registro previo. La segunda de ellas es “Participation” con una multiplicidad de cero a infinito, lo que indica que el evento puede no tener ninguna, una o infinitas participaciones.

Por último, dispone de una serie de operaciones relativas al evento, de entre las cuales destacamos:

- `findOwnedEvents()` necesita un token de acceso y permite obtener todos los eventos creados por el usuario propietario de dicho token.
- `claim()` permite reclamar un evento sin propietario, recibiendo como atributo el ID del evento que se desea reclamar y un token que se genera y proporciona únicamente en el momento de la creación, de esta forma nos aseguramos de que únicamente la persona que creó el evento puede reclamarlo y por lo tanto modificarlo, seleccionar una fecha definitiva, etc.

3.1.3. Clase “Participation”

Se encarga de mantener la información y proporcionar las operaciones necesarias

relativas a las participaciones de un evento, de forma que encontramos una serie de atributos visibles tales como el ID, nombre y apellidos, y otros con visibilidad privada como “participationToken”.

Además, dispone de tres relaciones: la primera con la clase “Event” tiene una multiplicidad de 1, lo que nos indica que la participación debe estar relacionada obligatoriamente con un único evento. La segunda con “Selection” que tiene una multiplicidad de uno a infinito, lo que nos indica que es necesario disponer de al menos una selección para que pueda existir una participación, y en el caso de actualizar las selecciones a una lista sin elementos, se eliminará la participación. Por último, tenemos una relación con la clase “User” con multiplicidad de cero a uno, lo que nos permite crear una participación sin necesidad de estar registrado o iniciar sesión.

No obstante, hay que tener en cuenta una serie de detalles para la creación de una participación en un evento. En primer lugar, es necesario que contenga al menos una selección válida (tal y como comentamos anteriormente), que la fecha de creación sea anterior a la de expiración del evento (si existe) y que se introduzca la contraseña del evento (si existe). En el caso de que se introduzca todos los datos correctamente y se haya proporcionado un token de acceso, se generará una instancia de participación relacionada con el usuario propietario del token mediante la relación “user”, esto permitirá realizar futuras modificaciones de la participación. Sin embargo, si no se ha proporcionado ningún token de acceso, se requerirá introducir al menos un nombre y no se usará la relación “user”, pero en su lugar se generará y proporcionará un token de participación que permitirá al usuario realizar futuras modificaciones sin necesidad de un registro previo o iniciar sesión.

3.1.4. Clase “Selection”

Se encarga de mantener la información y proporcionar las operaciones necesarias relativas a las selecciones realizadas en una participación, por lo que además de tener atributos tales como “comment”, que permite almacenar un comentario relativo a la selección y “period” que almacena un tipo de dato “Period” (detallado a continuación) para representar la selección, dispone de una relación con “Participation” de multiplicidad 1, siendo obligatorio estar relacionado con una participación, ya que no tendría sentido realizar una selección sin relación alguna.

Para crear una nueva selección, es necesario indicar la participación donde se almacenará esta selección y, en el caso de que la participación esté relacionada con un usuario, es necesario proporcionar un token de acceso válido para dicho usuario. Si la participación no está relacionada con ningún usuario, se debe proporcionar el token de participación que se proporcionó en el momento de la creación del mismo. Si por el contrario no se proporciona ninguno de estos tokens, no se permitirá la creación de esta nueva selección.

3.1.5. Tipo de dato "Period"

Se trata de un tipo de dato que permite representar un periodo de tiempo. Un periodo de tiempo se compone de un inicio y final, por lo que se podría componer de dos atributos de tipo Date indicando dicha información, sin embargo, debido a la forma en la que un ordenador trata las fechas, supone un problema principalmente en dos situaciones muy frecuentes:

- Si se desea seleccionar un día completo, la hora de inicio debe ser 00:00:00, aunque la hora final no puede ser las 00:00:00, ya que estaríamos indicando que el evento tiene una duración de 0 minutos o que acaba el día siguiente, por lo que debemos establecer que la hora final es a las 23:59:59 con 999 milésimas, lo que nos indica una duración de 1439,999 minutos en lugar de 1440.
- En el caso de querer especificar un evento cuya duración hace que la ficha final pertenezca al día siguiente, por ejemplo, comenzando a las 11 PM y acabando a las 2 AM del día siguiente, lo más natural para una persona es tratarlo como un único evento, sin embargo, dado que el ordenador trata las fechas por día, debe representarse como dos eventos, uno que comienza a las 23:00:00 y a acaba a las 23:59:59.999 y otro que comienza a las 00:00:00 del día siguiente y acaba a las 02:00:00. Una representación nada natural para el usuario.

Para solucionar este problema y hacer que la representación sea mucho más sencilla para el usuario, se ha decidido usar una estructura con dos atributos, el primero es "start" que almacena un tipo Date indicando la fecha y hora de inicio, y el segundo es

“duración”, un entero que indica la duración del evento en minutos (la máxima precisión permitida).

De esta forma, tanto el backend como frontend podrá calcular la fecha de finalización del evento, permitiendo realizar las operaciones necesarias, pero proporcionando al usuario una visión mucho más unificada de aquellos eventos que acaben el día siguiente o duren un día completo. Los ejemplos anteriormente comentados se representarían de la forma que indica la figura 9.

```
// Un día completo
period: {
  start: "2018-01-01T00:00:00.000Z",
  duration: 1440
}

// Comienza a las 11PM y finaliza a las 2AM del día siguiente
period: {
  start: "2018-01-01T23:00:00.000Z",
  duration: 180
}
```

Figura 9. Estructura “Period”

3.1.6. Tipo de dato “Day”

Es un tipo de dato definido únicamente para el atributo “days” de la clase “Event” con el objetivo de poder representar la disponibilidad de una forma mucho más precisa y permitiendo una mayor flexibilidad. Para ello se compone de dos atributos, el primero es “period” y almacena información sobre qué día se ha seleccionado y el tramo horario disponible, de esta forma conseguimos una mayor flexibilidad que las herramientas analizadas en el apartado 1.2 de este documento, ya que un usuario puede seleccionar cualquier tramo horario disponible dentro de este rango. Sin embargo, el segundo atributo, “blockedPeriods”, permite especificar la disponibilidad con más detalles, permitiendo bloquear tramos horarios dentro del tramo indicado por el atributo “period”.

Para ejemplificar el funcionamiento de “blockedPeriods”, supongamos que se desea realizar un evento donde se pueda seleccionar la disponibilidad desde las 10:00 hasta las 20:00, pero no queremos que los participantes puedan seleccionar desde las 14:00 hasta las 15:00 al ser la hora del almuerzo. El atributo “Day” se representaría de la forma que indica la figura 10.

```
day: {
  period: {
    start: "2018-01-01T10:00:00.000Z",
    duration: 600
  },
  blockedPeriods: [
    period: {
      start: "2018-01-01T14:00:00.000Z",
      duration: 60
    }
  ]
}
```

Figura 10. Estructura "Day"

3.2. Estructura de la base de datos

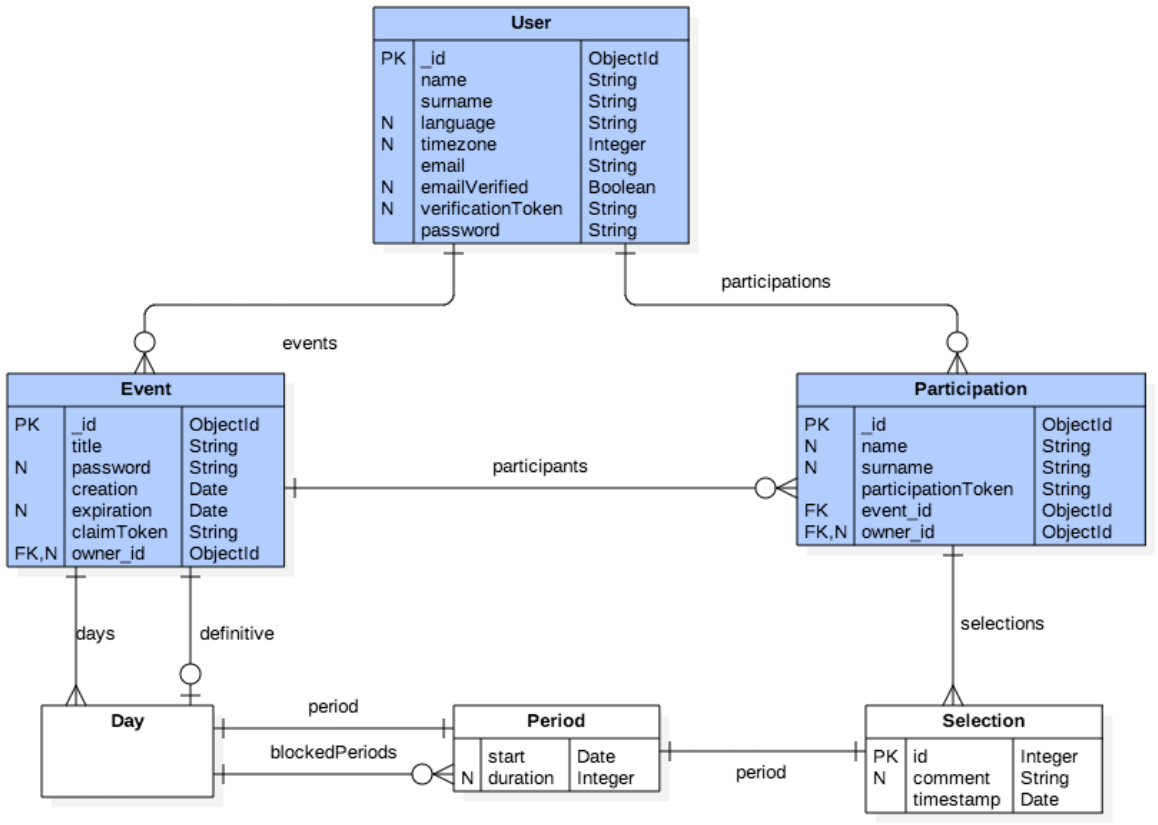


Figura 11. Diagrama entidad-relación UML

Una vez definida la estructura del backend mediante un diagrama de clases, es conveniente modelar el esquema de la base de datos, evitando así problemas de redundancia, permitiendo una mayor escalabilidad y facilitando la mantenibilidad. Sin embargo, dado que anteriormente únicamente hemos modelado bases de datos relacionales, se requirió un estudio previo de las herramientas de modelado de bases de datos no relacionales disponibles y tras probar distintas alternativa, se llegó a la

conclusión que no es necesario acudir a una herramienta diseñada exclusivamente para bases de datos específicas, MongoDB en nuestro caso, sino que se puede usar cualquiera siempre que permita representar todas las características del sistema que estamos utilizando, por lo que finalmente se decidió realizar el modelado en un diagrama entidad-relación (EDR) genérico mediante la misma herramienta con la que hemos modelado el diagrama de clases, StarUML, obteniendo como resultado el diagrama que se muestra en la figura 11.

En azul se muestra las entidades persistentes en la base de datos, mientras que las otras entidades nos permiten definir la estructura de algunos atributos de las entidades persistentes.

Dado que estamos usando una base de datos no relacional, las relaciones hacia otras entidades persistentes que se muestran en el esquema son tratadas como atributos de claves foráneas, de forma que nos permite simular una relación tradicional y podemos evitar la redundancia de datos. Sin embargo, las relaciones a las entidades no persistentes se tratan como un atributo, o un array en el caso de ser una relación uno a muchos, cuyo tipado corresponde a la entidad relacionada.

3.2.1. Entidad “User”

Consiste en una entidad persistente que almacena información relativa al usuario, cuya clave primaria es un ID generado automáticamente por la base de datos. Este ID consiste en un tipo de dato definido por MongoDB y denominado ObjectId, que consiste en un valor de 12 bytes cuyos 4 bytes más significativos representan los segundos en formato Unix, los siguientes 5 bytes representan un valor aleatorio y los últimos 3 bytes menos significativos pertenecen a un contador que comienza por un número aleatorio.

Además, contiene los atributos definidos por el diagrama de clases, donde “language”, “timezone”, “emailVerified” y “verificationToken” son valores opcionales, además de “events” y “participations” ya que son claves foráneas a las entidades “Event” y “Participation” respectivamente que permiten almacenar cero elementos.

3.2.2. Entidad “Event”

Consiste en una entidad persistente que almacena información relativa a un evento, cuya clave primaria consiste en una cadera de caracteres única y aleatoria con un tamaño mucho más reducido a los IDs generados automáticamente por MongoDB. En próximos apartados se detallará el algoritmo utilizado para generar los identificadores para los eventos.

En cuanto a los atributos, contiene los mismos definidos en el diagrama de clases, siendo “password”, “expiration” y “definitive” opcionales, además de las claves foráneas “participants”, que permite almacenar una lista vacía, y “owner_id”, para permitir así que un usuario no registrado pueda crear un evento.

Las relaciones “days” y “definitive” se tratan como un array y atributo respectivamente de tipo “Day”, el cual contiene dos atributos, “period” y “blockedPeriods”, que consisten en un atributo y array respectivamente de tipo “Period”, que a su vez consiste en un objeto con dos atributos, “start” (de tipo Date) y “duration” (de tipo Integer), cuyo propósito se ha explicado en el apartado anterior.

Finalmente, comentar que el atributo “days” es un array con multiplicidad uno a infinito, lo que nos indica que la entidad debe contener al menos un día para ser válida.

3.2.3. Entidad “Participation”

Consiste en una entidad persistente que almacena información relativa a una participación en un evento, cuya clave primaria consiste en un ID generado automáticamente por la base de datos MongoDB y cuyo funcionamiento hemos explicado anteriormente.

Esta entidad dispone de todos los atributos que definen el diagrama de clases, siendo “surname” opcional, y aunque “name” y “owner_id” se representen como atributos opcionales, en la práctica tan sólo uno de ellos debe serlo, de forma que si una participación no tiene un atributo “owner_id”, debe indicarse un nombre mediante el atributo “name” y viceversa.

Finalmente, tal y como se comentó anteriormente, la relación “selections” se trata como un array de objetos de tipo “Selection”, que se define con un ID como clave

primaria (en este caso se trata de un atributo con un contador de formato Integer, ya que no es necesario que sea difícil de predecir), un comentario opcional, un timestamp que guarde la fecha y hora de la creación y un periodo de tipo “Period” que indique el rango que se desea seleccionar.

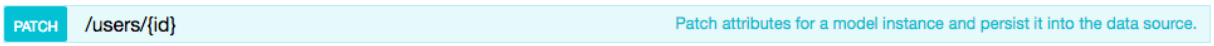
3.3. Interfaz y funcionamiento de la API REST

Dado que el backend de la aplicación no está destinado exclusivamente para servir como backend de un cliente web, sino que debe ser un sistema que permita la integración de cualquier futuro proyecto, es fundamental diseñar una API REST con una interfaz clara, sencilla, versátil y lo más importante, debe seguir unas buenas prácticas, más si tenemos en cuenta que estará abierta a terceros.

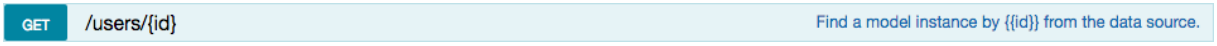
Para ello se realizó un estudio sobre cómo definir correctamente una API REST y se investigaron una multitud de servicios ya en funcionamiento. En la bibliografía de este documento se muestra las fuentes consultadas más importantes.

3.3.1. Interfaz para operaciones relativas al usuario (/users)

Permite crear un nuevo usuario, para ello es necesario proporcionar un objeto con los datos especificados previamente y se enviará un correo electrónico que contendrá un enlace para activar la cuenta, intercambiando un token de activación por la activación de la cuenta.



Permite modificar tan sólo aquellos atributos que se proporcionen del usuario cuyo ID corresponda con el especificado. Para ello es necesario un token de acceso que pertenezca al usuario que se desea modificar.



Permite obtener el usuario cuyo ID corresponda con el especificado

Permite modificar el usuario cuyo ID corresponda con el especificado, sustituyendo todos los atributos por los que se proporcionen en la petición. Para ello es necesario un token de acceso que pertenezca al usuario que se desea modificar.

DELETE /users/{id} Delete a model instance by {{id}} from the data source.

Permite eliminar el usuario cuyo ID corresponda con el especificado, para ello es necesario un token de acceso que pertenezca al usuario que se desea eliminar.

POST /users/{id}/replace Replace attributes for a model instance and persist it into the data source.

Tiene la misma funcionalidad que la petición PUT a /users/{id}, permitiendo modificar el usuario cuyo ID corresponda con el especificado. Para ello es necesario un token de acceso que pertenezca al usuario que se desea modificar.

POST /users/change-password Change a user's password.

Permite modificar la contraseña del usuario proporcionando la contraseña actual y la nueva.

GET /users/confirm Confirm a user registration with identity verification token.

Permite verificar un usuario mediante un token de verificación que se envía únicamente por correo en el momento de la creación, también debe especificarse el ID del usuario y una URL de retorno opcional.

POST /users/login Login a user with username/email and password.

Permite iniciar sesión, obteniendo un token de acceso en caso de proporcionar un email y contraseña válidos. Este token de acceso permitirá acceder a servicios restringidos únicamente para los usuarios registrados.

POST /users/logout Logout a user with access token.

Permite cerrar sesión, eliminando el token de acceso y por lo tanto no siendo válido para ninguna petición futura. Para ello es necesario proporcionar un token de acceso que pertenezca al usuario que desea cerrar sesión.

POST /users/reset Reset password for a user with email.

Permite enviar un mensaje de correo electrónico a la dirección especificada, donde se incluirá un enlace que permite cambiar la contraseña usando un token de acceso temporal.

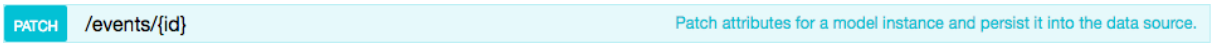
POST /users/reset-password Reset user's password via a password-reset token.

Permite cambiar la contraseña sin necesidad de especificar la contraseña actual, en su lugar se deberá especificar un token de acceso.

3.3.2. Interfaz para operaciones relativas al evento (/events)

Requiere proporcionar un token de acceso y permite obtener todos los eventos creados por el usuario propietario de dicho token.

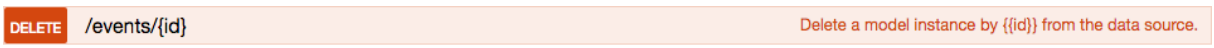
Permite a cualquier usuario, con token de acceso o sin él, crear un evento, para ello debe proporcionarse un objeto con los datos especificados anteriormente.



Permite modificar tan sólo aquellos atributos que se proporcionen del evento cuyo ID corresponda con el especificado. Para ello es necesario un token de acceso que pertenezca al usuario que creó el evento.

Permite obtener el evento cuyo ID corresponda con el especificado.

Permite modificar el evento cuyo ID corresponda con el especificado, sustituyendo todos los atributos por los que se proporcionen en la petición. Para ello es necesario un token de acceso que pertenezca al usuario que creó el evento.



Permite eliminar el evento cuyo ID corresponda con el especificado, para ello es necesario un token de acceso que pertenezca al usuario que creó el evento.

Permite reclamar un evento, para ello es necesario el token de acceso del usuario al que se le asignará el evento, y un token de reclamación que se proporciona en el momento de la creación del evento.

Permite obtener todas las participaciones del evento cuyo ID corresponda con el especificado.

Permite crear una participación para el evento cuyo ID corresponda con el especificado. Para ello debe proporcionarse un objeto con los datos especificados anteriormente y cumplir con los requisitos (debe contener selecciones válidas, no ser

posterior a la fecha de expiración del evento e indicar la contraseña en el caso de estar protegido). En el caso de no proporcionar un token de acceso, se generará un token de participación que será necesario para realizar cualquier futura modificación.

GET /events/{id}/participations/{part_id} Find the specified participation in the event.

Permite obtener una participación específica del evento cuyo ID corresponda con el especificado. También se proporcionará todas las selecciones relativas a la participación.

POST /events/{id}/participations/{part_id}/selections Create a selection for a participation.

Permite establecer las selecciones de la participación especificada en el evento cuyo ID corresponda con el especificado. Para ello es necesario proporcionar el token de acceso del usuario que creó la participación o el token de participación que se proporcionó en el momento de la creación.

POST /events/{id}/replace Replace attributes for a model instance and persist it into the data source.

Tiene la misma funcionalidad que la petición PUT a /events/{id}, permitiendo modificar el evento cuyo ID corresponda con el especificado. Para ello es necesario un token de acceso que pertenezca al usuario que creó el evento.

3.4. Estructura del código fuente

Una vez que se disponga de los diagramas especificando el modelo de la aplicación y la estructura de la base de datos, se puede comenzar con la implementación, sin embargo, dado que se estaba usando un framework que no habríamos utilizado anteriormente, antes de comenzar se realizó un estudio sobre cuál sería la forma más conveniente de organizar el código fuente, además de inspeccionar numerosos proyectos de código libre usando la misma tecnología. Finalmente se decidió utilizar la estructura que se muestra en la figura 11.

Como se puede observar, disponemos de un directorio dedicado exclusivamente al cliente de la aplicación, de forma que podamos tener ambos proyectos separados. A continuación, tenemos un directorio llamado “common” en el que se almacena los modelos que requiere el framework LoopBack. Estos consisten en archivos JSON que se encargan únicamente de definir la estructura del modelo y archivos JavaScript que dotan de funcionalidad a cada una de las operaciones definidas el modelo.

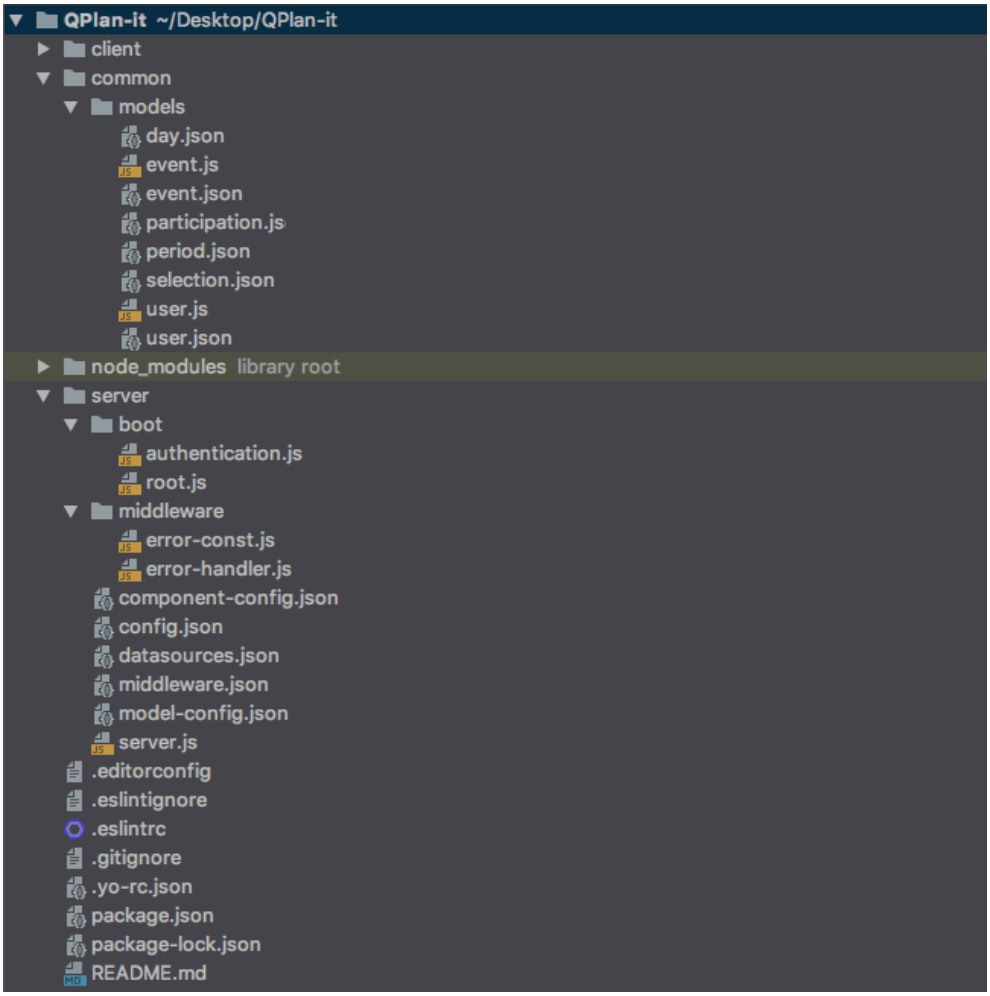


Figura 12. Estructura del código fuente del backend

Posteriormente tenemos el directorio “server” en el que se almacena los archivos necesarios para el correcto funcionamiento del servidor:

- Directorio “boot”: Archivos generados por el LoopBack para el inicio.
- Directorio “middleware”: Almacena el middleware encargado de la gestión de errores que detallaremos posteriormente.
- Archivo “component-config.json”: Configuración del explorador para la API.
- Archivo “config.json”: Configuración del servidor, indicando parámetros tales como el la URL de la API, dirección del servidor, puerto, etc.
- Archivo “datasources.json”: Configuración de la conexión con la base de datos.
- Archivo “middleware.json”: Especificación de los middlewares instalados en el servidor.
- Archivo “model-config.json”: Especificación y configuración de los modelos disponibles en el servidor.

Por último, además del archivo “package.json” que especifica la configuración y dependencias del proyecto Node.js, tenemos una serie de archivos que configuran ciertos aspectos del proyecto que no son relevantes para la descripción en este documento.

3.5. Detalles de implementación

Para facilitar la lectura del documento, no se comentará todo el código fuente del proyecto, pero consideramos relevante detallar cómo se han solventado ciertos problemas a la hora de la implementación, tales como la generación de ID únicos para eventos, la gestión de fechas y zonas horarias, el control de acceso, gestión de errores, etc.

3.5.1 Generación de ID únicos para eventos

Tal y como se ha comentado anteriormente, la entidad “Event” usa un ID distinto al generado por la base de datos MongoDB para usar un algoritmo propio en su lugar, esto es debido a que los identificadores generados por MongoDB tienen una longitud de 12 bytes o 24 caracteres, sin embargo, dado que necesitamos generar enlaces lo más cortos posibles para que sea fácil de compartir, necesitamos un identificador con una longitud sustancialmente menor, de forma que hemos implementado un algoritmo para generar identificadores únicos para un número infinito de eventos y que sean difíciles de predecir.

```
async function generateUniqueId() {
  let uniqueId, unique = false, count = 0;

  while(!unique) {
    if(count === 5) {
      config.eventIdLength = config.eventIdLength + 1;
      let file = editJsonFile(`${__dirname}/../../server/config.json`, {});
      file.set("eventIdLength", config.eventIdLength);
      file.save();
      count = 0;
    }
    uniqueId = new hashids(shortid.generate(), config.eventIdLength).encode(1);
    await model.findById(uniqueId).then(event => unique = !event);
    count++;
  }

  return uniqueId;
}
```

Figura 13. Algoritmo para generación de IDs únicos

Para ello se ha utilizado la librería “shortid” (https://github.com/dylang/shortid) que permite generar identificadores con una gran entropía y un tamaño de entre 7 a 14 caracteres. Posteriormente se ha utilizado una librería que permite generar identificadores de una longitud específica a partir de una salt y un número, la librería en cuestión se llama “hashids” (https://github.com/ivanakimov/hashids.js), por lo que se usa como número el 1 y el ID generado por la librería “shortid” como salt, además le indicamos la longitud que especifica el archivo de configuración del servidor (por defecto 6).

En este punto ya tenemos un identificador alfanumérico, sensible a mayúsculas y con una longitud de 6 caracteres (alfabeto de 62 caracteres), por lo que tenemos la capacidad de generar 62^6 = 56.800.235.584 identificadores. Además, teniendo en cuenta la entropía de la generación del identificador, la probabilidad de generar un ID repetido es mínima, pese a ello, se ha implementado un mecanismo para poder generar infinitos identificadores únicos. Para ello se realiza una consulta a la base de datos y se comprueba si ya existe otro evento con el mismo ID. En el caso de que exista, se genera uno nuevo, sin embargo, si ha llegado un punto en el que se ha generado 5 identificadores y todos ellos ya existen (lo que probabilísticamente significa que estamos llegando al límite del número máximo de IDs que podemos generar), se modifica el archivo de configuración para aumentar la longitud del identificador, de forma que la nueva longitud sería 7 caracteres, pudiendo generar 62^7 = 3.521.614.606.208 nuevos identificadores.

De esta forma podemos generar unas URLs tales como qplan.it/WXnxeA sin preocuparnos por tener solapamientos en el ID del evento.

3.5.2 Gestión de fechas y zonas horarias

Debido a que el planeta está dividido por un total de 27 zonas horarias y estamos trabajando con planificaciones horarias, debemos tener en cuenta este detalle, de forma que si se crea un evento en un lugar cuya zona horaria es UTC+1, al visitarlo desde otro lugar con zona horaria UTC, debe mostrarse que comienza una hora antes en comparación con la primera ubicación.

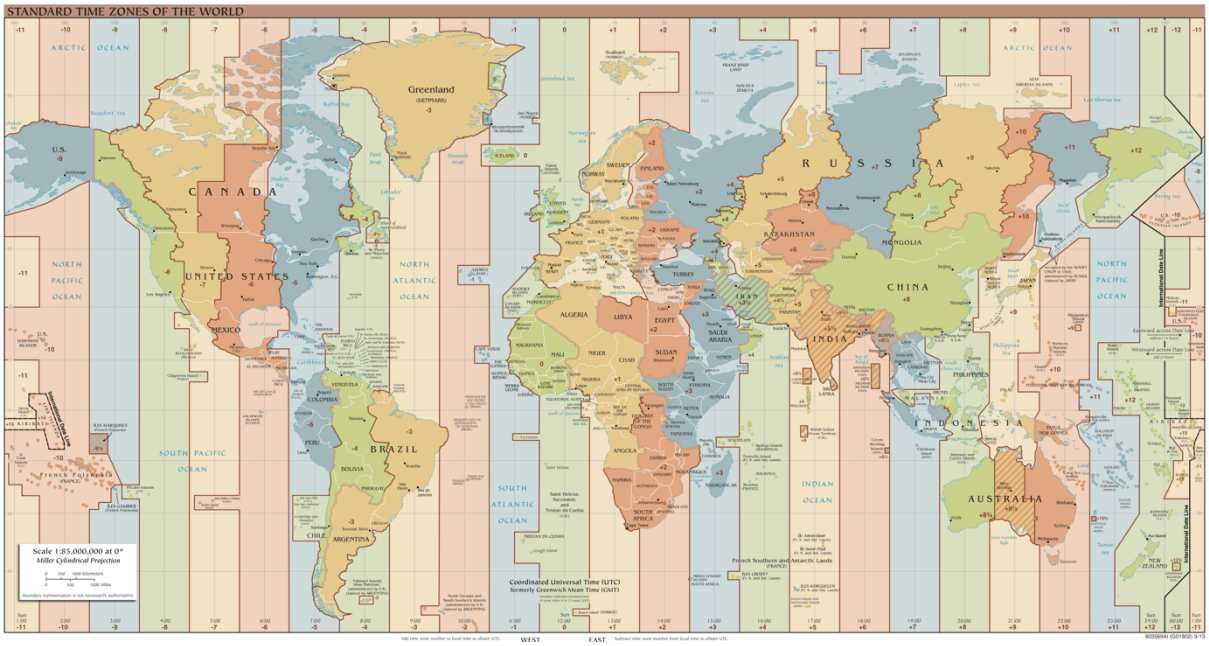


Figura 14. Mapa de zonas horarias (https://en.wikipedia.org/wiki/Time_zone)

Para solucionar este problema, toda fecha se traduce a una hora UTC para operar con ella y guardarla en la base de datos, permitiendo al cliente realizar las transformaciones necesarias para mostrar la hora en la zona horaria correspondiente.

Para realizar esta transformación necesitamos conocer la zona horaria del lugar donde se hizo la petición, esto es posible gracias a que el cliente suele estar configurado con la zona horaria correcta y nos proporciona la fecha en formato ISO-8601, donde se especifica si la hora se encuentra en formato UTC (añadiendo una Z al final de la misma) o se encuentra en una configuración local (en cuyo caso se especifica la zona horaria al final).

3.5.3. Control de acceso y validación de datos

La aplicación tiene un control de acceso gestionado por el framework LoopBack, de forma que en la mayoría de ocasiones tan sólo debemos definir quién tiene acceso a cada servicio proporcionado por la API mediante la configuración del modelo, tal y como se puede ver en la figura 15.

En ese ejemplo estamos denegado el acceso de todas las operaciones a todo el mundo para posteriormente permitirles el acceso a la función “create”. De esta forma podemos denegar completamente el acceso a una función si no cumple con los requisitos.

Figura 15. Control de acceso mediante LoopBack

Sin embargo, hay otras circunstancias donde el control de acceso es más específico, por ejemplo, debemos denegar el acceso a la creación de una participación si el usuario no ha iniciado sesión (no se proporciona un token de acceso) y no se proporciona un token de participación. Para ello debemos recurrir al código fuente, donde controlamos específicamente ese caso, tal y como se muestra en la figura 16.

```
if((!part.ownerId || !options.accessToken || part.ownerId.toString() !== options.accessToken.userId.toString()) &&
  (!part.participationToken || !partToken || part.participationToken !== partToken)) {
  throw ErrorConst.Error(ErrorConst.AUTHORIZATION_REQUIRED)
}
```

Figura 16. Control de acceso personalizado

En cuanto a la validación de datos, nos encontramos en la misma circunstancia, el framework de LoopBack nos permite definir qué atributos debe tener un modelo y cuáles de ellos son obligatorios, rechazando todas aquellas peticiones que no cumplan con lo especificado. Esto podemos configurarlo desde el archivo JSON que usamos para la definición del modelo, tal y como se puede ver en la figura 17.

Figura 17. Validación de atributos mediante LoopBack

Por contra, no nos permite definir situaciones más complejas como detectar cuándo un conjunto de días es válido, por lo que debemos especificarlo en el archivo JavaScript. A continuación, a modo de ejemplo, se muestra la función que se encarga de validar un conjunto de días.

```
function ensureValidDays(days) {
  let startDate, endDate, dayTime, dayTimes = [];

  if(!days || days.length === 0) {
    throw ErrorConst.Error(ErrorConst.DAYS_REQUIRED);
  }

  days.forEach(day => {
    if(!day.period || !day.period.start || isNaN(day.period.start.getTime())) {
      throw ErrorConst.Error(ErrorConst.PERIOD_REQUIRED);
    }

    startDate = moment(day.period.start);
    dayTime = moment(startDate).startOf('day').toDate().getTime();

    if(!dayTimes.includes(dayTime)) {
      dayTimes.push(dayTime);
    } else {
      throw ErrorConst.Error(ErrorConst.DAYS_NOT_UNIQUE);
    }

    if(!day.period.duration || day.period.duration <= 0) {
      day.period.duration = Math.floor(moment(startDate).endOf('day').diff(startDate) / 60000) + 1;
    } else if(day.period.duration > 1440) {
      day.period.duration = 1440;
    }
    endDate = moment(startDate).add(day.period.duration, 'm');
    if(day.period.duration === 1440 && endDate.hours() === 0 && endDate.minutes() === 0) {
      endDate = moment(startDate).endOf('day');
    }

    if(day.blockedPeriods) {
      day.blockedPeriods.forEach(period => {
        if(!period.start || isNaN(period.start.getTime())) {
          throw ErrorConst.Error(ErrorConst.PERIOD_REQUIRED);
        }

        let bpStart = moment(startDate).hours(period.start.getHours()).minutes(period.start.getMinutes());
        let minutesToEnd = Math.floor(endDate.diff(bpStart) / 60000) % 1440;

        if(!period.duration || period.duration <= 0 || period.duration > minutesToEnd) {
          period.duration = minutesToEnd;
        }
      });
    }
  });
}
```

Figura 18. Validación de atributos personalizada

Finalmente, indicar que a pesar que consideramos positivo que el cliente incorpore una validación de datos, de forma que no sólo se consigue un aumento de velocidad, sino también se reduce el número de peticiones al servidor y por lo tanto se reduce su carga, consideramos fundamental que las validaciones también se hagan por parte del servidor para evitar que se registren peticiones inválidas y obtener un mayor nivel de seguridad. El control de acceso, por el contrario, es obligatorio gestionarlo desde el backend, ya que el cliente se puede modificar fácilmente.

3.5.4. Generación de tokens y cifrado de contraseñas

Un token consiste en una cadera de caracteres aleatoria con una probabilidad de colisión ínfima que nos permite identificar un usuario sin necesidad de procesar o almacenar su contraseña.

Afortunadamente los tokens de acceso al iniciar sesión y tokens de acceso temporal los genera LoopBack, el framework que estamos utilizando, sin embargo, hay

ocasiones en las que necesitamos generar nuestro propio token, como el "claimToken", usado para reclamar un evento, el "verificationToken", usado para verificar la cuenta de un usuario, y el "participationToken", usado para permitir realizar modificaciones a una participación sin necesidad de iniciar sesión.

Para ello, en lugar de especificar un algoritmo, hemos acudido a una librería ampliamente probada llamada "rand-token" (https://github.com/sehrope/node-rand-token), de forma que podamos generar tokens más seguros de una forma rápida. Esta librería nos permite indicar la longitud del token generado, por lo que hemos seleccionado una longitud de 64 caracteres, obteniendo así un token con una probabilidad de colisión muy baja.

En cuanto al cifrado de las contraseñas, o más específicamente, la generación de un hash de la contraseña, nos encontramos en la misma circunstancia que la generación de tokens, la librería LoopBack se encarga de generar un hash de forma automática, aun así, dado que permitimos establecer una contraseña de participación en un evento, nos parece adecuado guardarla en la base de datos de una forma segura, por lo que hemos acudido a la implementación en Node.js de una librería ampliamente conocida, "bcrypt" (https://github.com/kelektiv/node.bcrypt.js), y generado un hash de la contraseña usando un salt aleatorio que nos permite generar la misma librería. De esta forma, para validar si la contraseña es correcta, podemos generar un hash de la misma y comparar si ambos hashes coinciden.

Por último, indicar que se ha implementado un sistema de comprobación de fortaleza de una contraseña, permitiendo establecerla únicamente si cumple con los requisitos de tener mayúsculas, minúsculas y al menos un número. En caso contrario se responderá con un error indicando que la contraseña debe ser más segura.

3.5.5. Gestión de errores

Si realizamos una petición a la API REST y la respuesta no es satisfactoria, consideramos fundamental recibir una información clara y precisa del problema que ha ocurrido, sin embargo, el framework de LoopBack, aunque contiene una gestión de errores, consideramos que es ampliamente mejorable, ya que no mantiene una lista unificada con los códigos de error y una descripción de lo sucedido, además de no

proporcionar ningún código de error en ciertas ocasiones, por lo que hemos decidido implementar un middleware que se encargue de gestionar los errores.

Este middleware tiene tres funcionalidades principalmente: la primera de ellas se encarga de detectar los errores más comunes del sistema de gestión de errores de LoopBack y los traduce a un sistema propio que mantiene una lista unificada de los códigos de error, su descripción y código http de respuesta. La segunda función es permitir lanzar errores de esta lista en cualquier parte del código, facilitando así las duplicidades de códigos y mensajes de error. Y, por último, una característica más simple pero necesaria, en caso de ocurrir un error desconocido se muestra usando el mismo sistema de errores, en lugar de producir mensajes inesperados.

Este sistema de errores permite responder con el código http correspondiente y envía como contenido un mensaje que siempre se ajusta a la estructura mostrada en la figura 19 y donde “code”, “statusCode” y “message” varía según el error.

Figura 19. Estructura de los errores

4. Frontend de la aplicación

Como se ha comentado en la sección 1.4, para la implementación del frontend se ha utilizado React, permitiendo crear una aplicación JavaScript que se ejecuta en el navegador del cliente y que, por lo tanto, consume menos recursos del servidor, además de mejorar la experiencia de usuario.

Dado que la característica diferenciadora principal de esta herramienta con respecto a las ya existentes en el mercado es el cumplimiento de los requisitos indicados en el capítulo 1.1 de este documento, sencillez, flexibilidad y escalabilidad, es fundamental desarrollar un frontend acorde a estas características, por lo que no sólo se dedicó una cantidad considerable de tiempo en estudiar las distintas posibilidades de diseño, maquetarlas y encuestar a personas cercanas para detectar posibles carencias, sino que es la implementación es iteración que más tiempo ha consumido del proyecto.

A continuación, se especificará los detalles del desarrollo del frontend, como patrones de diseño implementados, estructura del código fuente y otros.

4.1. Patrón de diseño

Antes de comenzar la implementación en la tecnología seleccionada, se estudió minuciosamente qué patrones de diseño existían para el tipo de proyecto que queremos desarrollar y cuál se ajusta más a nuestras necesidades.

En el ámbito de las páginas o aplicaciones web, en un principio no era necesario aplicar ningún patrón de diseño, debido a la simplicidad de las mismas, tratándose en su mayoría de páginas desarrolladas exclusivamente en HTML, un lenguaje de marcas de hipertexto, con enlaces que permitían la navegación de una página a otra. Sin embargo, conforme la web fue creciendo, surgieron proyectos cada vez más ambiciosos y más grandes, por lo que empezó a ser necesario el uso de algún patrón de diseño para realizar el frontend, de forma que se empezó a utilizar ampliamente el modelo vista-controlador (MVC), que consiste en una separación de la lógica, o modelo, de la aplicación y la vista, creando un controlador que se encargue de interactuar con el modelo y aplicar los cambios a la vista.

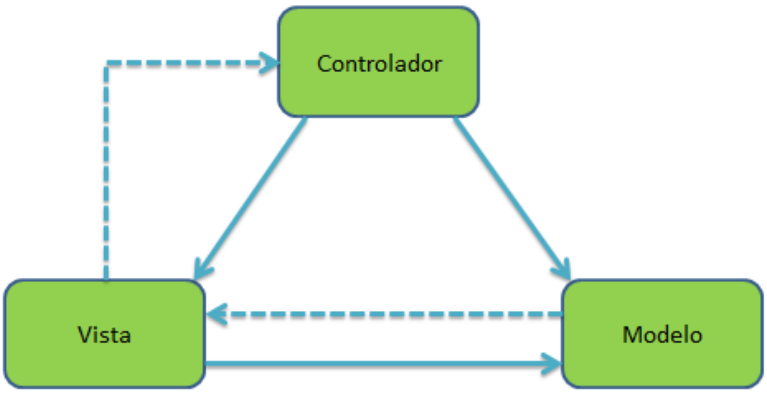


Figura 20. Esquema del modelo vista-controlador (https://www.oscarblancarteblog.com/2014/07/21/patron-de-diseno-modelo-vista-controlador-mvc)

Para facilitar aún más el desarrollo de clientes frontend surgieron frameworks como Angular, Blackbone o Ember, todos ellos usando este patrón de diseño, de forma que se podía conseguir aplicaciones web mucho más completas, facilitando enormemente la mantenibilidad y escalabilidad.

No obstante, a medida que las aplicaciones web seguían creciendo y empezaba a ser compleja la depuración y rastreo de errores cuando se tiene una comunicación bidireccional entre los modelos y controladores, surgía la necesidad de plantear un nuevo patrón de diseño.

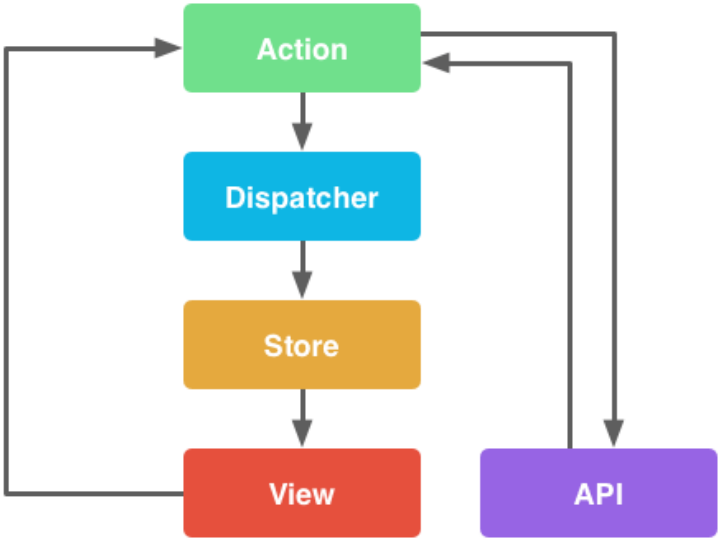


Figura 21. Esquema del modelo Flux (https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture)

De esta forma, y de mano de Facebook, nació el patrón de diseño Flux, una arquitectura en la que el flujo de datos es unidireccional, donde la vista genera una acción que permite modificar un "Store", que mantiene los datos y estado de la aplicación. Esto se hace mediante un "Dispatcher", siendo el único que puede modificar el "Store" y provocando que la vista se actualice. Este comportamiento se puede observar claramente en el esquema de la figura 21.

Este patrón de diseño nos pareció más adecuado para la implementación del frontend, sin embargo, en lugar de usar Flux, nos decantamos por Redux, una librería que implementa el patrón de diseño Flux e incorpora ciertas variaciones que consideramos muy positivas.

En primer lugar, Redux permite desacoplar el estado los componentes de la vista para usar un estado global de la aplicación que se encarga de mantener los datos que se han obtenido a partir de la API o generados mediante la interacción del usuario con la vista. Este estado global es el Store del patrón de diseño anteriormente comentado, aunque si bien, en Flux se propone el uso de varios Stores, Redux sólo utiliza uno.

En Redux se introduce una serie se conceptos cuya comprensión es necesaria para comprender el flujo de datos desde que se realiza una acción hasta que se refleja en la vista.

- **Operations:** Son el conjunto de operaciones de alto nivel que se llaman desde la vista, generando distintas acciones. Por ejemplo, una función de login permite generar tres acciones distintas, un LoginRequest, LoginSuccess o LoginFail.
- **Types:** Se trata de una serie de constantes que almacenan una cadena de texto descriptiva de la acción que se está produciendo.
- **Actions:** Son el conjunto de acciones que deben proporcionarse al Dispatcher para que éste se encargue de modificar el Store correctamente.
- **Reducers:** Son funciones que reciben dos parámetros, el estado inicial y una acción (la cual siempre contiene un type y puede contener un payload de datos), y definen las modificaciones que deben realizarse al estado según el tipo de la acción.

Para ejemplificar el flujo de datos, supongamos que un usuario hace clic en “Iniciar sesión” tras introducir sus credenciales. Esto provocaría que la vista llame a la operación de login, donde se generaría una acción de LoginRequest que contendría como type la cadena de texto “auth/LOGIN_REQ” y ningún payload. Posteriormente se llama al Dispatcher proporcionando esa acción y éste se encargaría de llamar al Reducer, el cual modifica el booleano “loading” del estado a “true”, provocando que la vista se actualice para mostrar un indicador de carga. En el momento que se reciba una respuesta por parte del servidor, ocurriría el mismo proceso, pero con acciones distintas.

Finalmente, detallaremos la estructura del Store que se ha implementado, siendo los objetos principales los que se muestran en la figura 22.

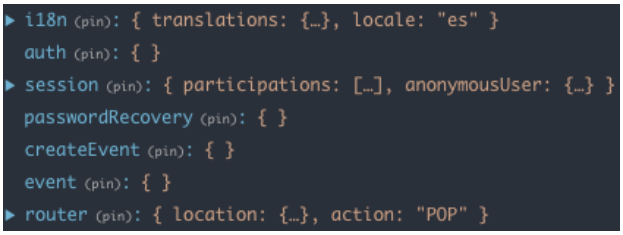


Figura 22. Estructura del Store

- **i18n:** Mantiene información relativa al idioma de la aplicación
- **auth:** Mantiene información relativa a la autentificación (si el panel está abierto, está cargando, etc.)
- **session:** Mantiene información relativa a la sesión (token de inicio de sesión, las participaciones realizadas, etc.)
- **passwordRecovery:** Mantiene información relativa a la recuperación de contraseña (si se está procesando la petición, los errores ocurridos, etc.)
- **createEvent:** Mantiene información relativa a la creación del evento (título, días seleccionados, etc.) permitiendo continuar con la creación en todo momento.
- **event:** Mantiene información relativa al evento visitado (título, participaciones, etc.)
- **router:** Mantiene información relativa al enrutador, permitiendo tener una visión global de la ruta de la aplicación.

4.2. Auto detección de idioma y traducción

Dado que la aplicación será accesible desde cualquier lugar, necesitamos traducir cada texto al idioma correspondiente, por lo que tras realizar un estudio las distintas herramientas y frameworks disponibles para realizar esta tarea, finalmente se ha decidido utilizar “react-redux-i18n” (https://github.com/artisavotins/react-redux-i18n), una librería que permite aprovechar las ventajas que nos proporciona Redux, comentado en el apartado anterior, de forma que podamos tener una visión del idioma de la aplicación de forma global.

Esta librería se configura en el Store de la aplicación y permite establecer un archivo donde se encontrará un objeto JSON con un conjunto de claves y valores que especifique la traducción para cada idioma definido. En la figura 23 se muestra una porción de este archivo donde se puede ver la estructura del objeto definido.

Figura 23. Archivo de traducción de idiomas

Sin embargo, esta librería no nos proporciona una traducción automática según el idioma del navegador, por lo que hemos desarrollado un mecanismo de detección de idioma apoyándonos en la librería “detect-browser-language”, cuyo repositorio se puede encontrar en https://github.com/bukinoshita/detect-browser-language.

```
let lang = detectBrowserLanguage() || "en";
if(lang.length > 2) {
  lang = lang.split('-')[0].split('_')[0];
}
lang = lang.toLowerCase();
store.dispatch(setLocale(Object.keys(i18n).indexOf(lang) !== -1 ? lang : "en"));
```

Figura 24. Detección automática de idioma

Como se puede observar, se realiza una llamada a una función de esta librería que permite obtener el idioma configurado en el navegador para posteriormente obtener el identificador de idioma en formato ISO 639-1, que consiste en un código de dos letras identificativo para el idioma. En el caso de no detectar el idioma correctamente o no existir una traducción para el idioma deseado, se mostrará en inglés.

A continuación, se realiza una llamada al Dispatch con la acción correspondiente que permite establecer el idioma de la aplicación en el Store.

4.3. Estructura del código fuente

De cara a futuras modificaciones donde puedan incorporarse terceros desarrolladores, consideramos fundamental no sólo mantener un código legible y usar buenas prácticas, sino que también debemos mantener una estructura de los ficheros fuente que permita identificar cada componente de la aplicación de una forma sencilla. Para ello se ha realizado un estudio sobre cómo sería más adecuado estructurar el proyecto, analizando el código fuente de diferentes proyectos reales en producción (los proyectos consultados pueden observarse en la bibliografía).

Finalmente se ha decidido utilizar la especificación “re-ducks”, cuyo repositorio se puede encontrar en https://github.com/alexnm/re-ducks. Éste se basa en la especificación “ducks” (https://github.com/erikras/ducks-modular-redux) donde se indica cómo debe estructurarse una aplicación usando Redux para que sea fácilmente legible y mantenible, sin embargo, a pesar de que la estructura cumple con su objetivo, tiene la carencia de usar un único archivo, por lo que, en proyectos a gran escala, puede suponer tener un archivo demasiado grande y que dificulte su lectura. Este detalle se soluciona en la especificación re-ducks, el cual propone separar cada característica de la aplicación en directorios distintos que contengan cada una de las características definidas en la especificación ducks (actions.js, operations.js…).

Esa forma de estructurar el proyecto nos pareció la más adecuada, ya que usualmente las modificaciones suelen realizarse en una característica concreta al mismo tiempo (dos a lo sumo), de forma que podemos navegar hacia el directorio que incluye nuestra característica y encontrar todo lo que necesitamos en el mismo sitio. En la figura 25 puede observarse cómo se ha desarrollado la aplicación siguiendo esta especificación, de forma que coincide con los objetos principales de nuestro Store comentados anteriormente.

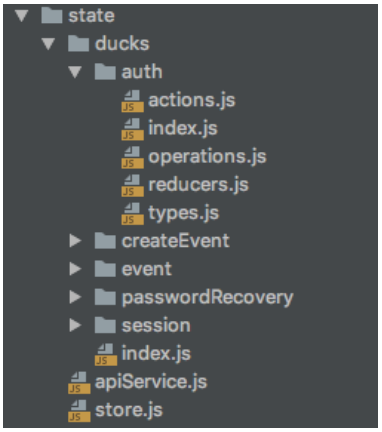


Figura 25. Estructura del código fuente del Store

Cada directorio que especifica una característica (auth, createEvent, event…) contiene un fichero para cada concepto del patrón Redux, los cuales se comentaron anteriormente, aunque sólo se ha expandido el primero de ellos para no adjuntar una figura con información redundante.

Además, puede verse que sólo se define un Store en la raíz del directorio “state” que reúne todos los ducks definidos en su carpeta correspondiente, además de un archivo “apiService.js” que sirve de interfaz entre una acción y la API REST proporcionada por el backend.

Finalmente, puede observarse la estructura del proyecto completo en la figura 26, donde la raíz tan sólo contiene el directorio “public”, donde se reúne todo el contenido que debe ser público y no necesita ser ejecutado como aplicación (tales como imágenes, el favicon, manifiesto, etc.), el directorio “src” donde se encuentra todo el código fuente de la aplicación y cuya estructura comentaremos a continuación, y otros archivos necesarios para la ejecución del servidor Node.js.

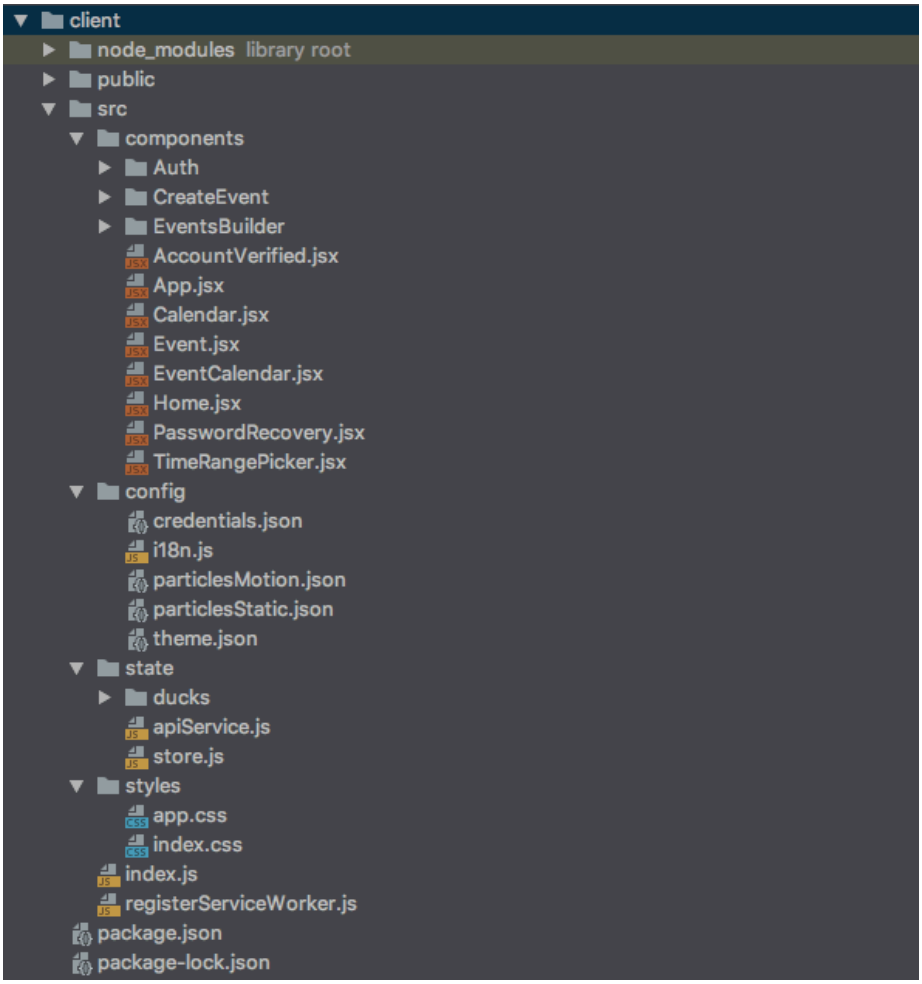


Figura 26. Estructura del código fuente del frontend

Dentro del directorio del código fuente ("src") puede observarse el siguiente contenido:

- **components:** Recopila todos los componentes de React.
- **config:** Recopila todos los archivos de configuración, desde las credenciales que almacenan las claves para servicios como re-captcha, hasta la configuración del tema de la aplicación.
- **state:** Directorio detallado anteriormente donde se recopilan todos los archivos relativos al estado de la aplicación.
- **styles:** Recopila todos los archivos CSS que especifican el diseño de la aplicación web.
- **index.js:** Archivo principal que define el componente raíz desde el que se extiende los demás componentes.
- **registerServiceWorker.js:** Archivo que permite almacenar la aplicación en el caché del navegador.

4.4. Enrutador de la aplicación

Anteriormente se ha comentado las ventajas de una aplicación web JavaScript que se ejecuta en el navegador del cliente, sin embargo, esto supone un nuevo reto con respecto al modelo de páginas web tradicionales, y es la navegación mediante URLs ya que, al no tratarse de distintas páginas web relacionadas por enlaces, sino una única aplicación, no podemos usar el sistema de enlaces tradicional.

Para solucionar este problema se usa una solución proporcionada por la librería "react-router" (https://github.com/ReactTraining/react-router), la cual nos permite simular el sistema de enlaces tradicional, facilitando navegar hacia una URL mediante JavaScript, establecer un estado de la aplicación dependiendo de la ruta en la que nos encontremos y cargar un componente distinto para cada ruta entre muchas otras características.

```
<Switch>
  <Route exact path="/" component={Home} />
  <Route exact path="/login" component={ () => this.renderLogin() } />
  <Route exact path="/create_event" component={ () => this.renderCreate() } />
  <Route exact path="/password_recovery" component={PasswordRecovery} />
  <Route exact path="/account_verified" component={AccountVerified} />
  <Route exact path="/:eventId" component={Event} />
</Switch>
```

Figura 27. Definición de rutas para el frontend

Como puede observarse, se ha definido una serie de rutas con un tamaño menor a 6 (longitud por defecto de los IDs de los eventos) o muy superior a éste, de forma que, si no se trata de ninguna de estas rutas predefinidas, podamos interpretar que el primer parámetro de la ruta se tata del ID de un evento sin temor a que un evento pueda coincidir con alguna de las rutas especificadas. Esto nos permite generar URLs para los eventos lo más pequeño posible para facilitar el que se pueda compartir fácilmente.

Además, nos gustaría indicar que, si hacemos clic en el botón de inicio de sesión desde la página principal, se navega a la ruta /login. De esta forma podemos compartir un enlace que permita abrir automáticamente esa ventana, sin embargo, si el mismo botón se presiona desde otra URL, se abre la misma ventana, pero no navega hacia ninguna ruta para no perder la dirección en la que nos encontramos.

4.5. Detalles de la implementación

La aplicación se ha desarrollado prestando una minuciosa atención a cada detalle, por lo que en este apartado se comentarán los detalles de implementación más relevantes. En primer lugar, nos gustaría indicar que se pretende usar la misma aplicación web como aplicación nativa de Android o iOS usando herramientas como Cordova (https://cordova.apache.org/), por lo que desde un principio de ha diseñado siguiendo unos principios de Responsive Design, lo que permite adaptar la interfaz gráfica de la aplicación web a distintos tamaños de pantalla, permitiendo usarse en dispositivos móviles. Además, se detecta mediante la librería "react-device-detect" (https://github.com/duskload/react-device-detect) si se está utilizando un dispositivo móvil para, en ese caso, modificar la configuración del elemento visual Particles.js (https://github.com/VincentGarreau/particles.js/) usado en la cabecera de la aplicación web y evitar el movimiento de partículas, de forma que no se utilicen los recursos necesarios para este propósito, ya que por lo general un dispositivo móvil dispone de menos recursos.

A continuación, se comentará algunos detalles de implementación relevantes en distintas áreas tales como la autentificación, creación del evento, etc.

4.5.1. Autentificación y registro de usuario

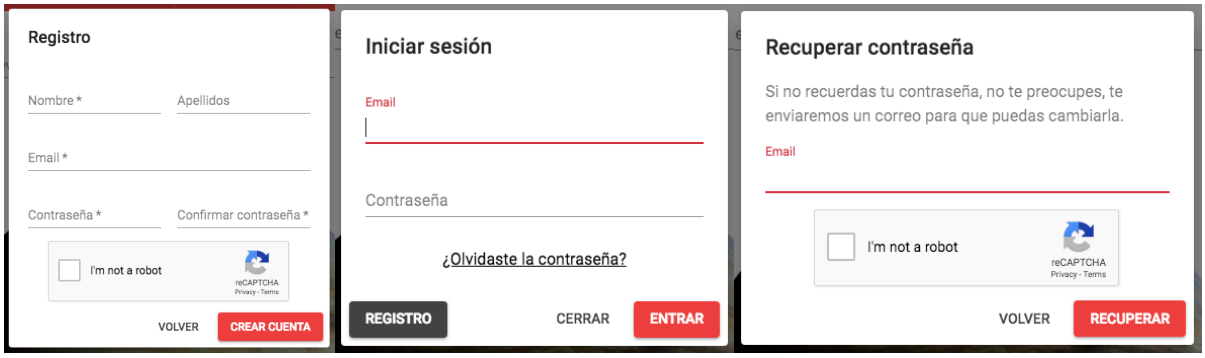


Figura 28. Ventana emergente de autentificación

Hemos decidido implementar todas las características relacionadas con la autentificación en una ventana emergente en lugar de una página completa, de forma que podamos iniciar sesión fácilmente sin necesidad de salir de la creación, modificación o participación en un evento. Para ello es necesario pulsar sobre el botón de iniciar sesión o navegar hace la ruta /login, mostrándose la ventana emergente y

desde la cual podemos intercambiar entre los tres modos disponibles, registro, inicio de sesión y recuperación de contraseña, tal y como se muestra en la figura 28.

Como es de esperar, las consultas se realizan de forma asíncrona, recibiendo información tanto en caso de fallo como de éxito siendo. A continuación, se muestra algunos de ellos.

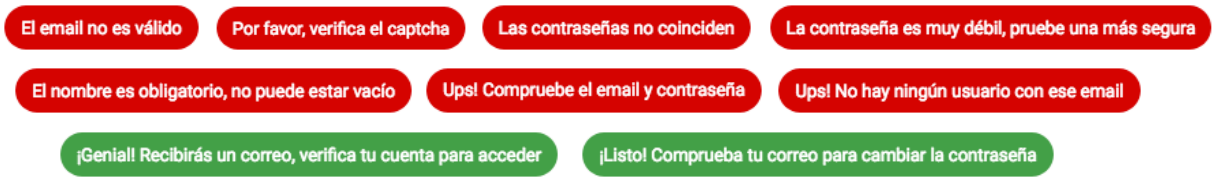


Figura 29. Mensajes de error y éxito en autentificación

En el caso de realizar un registro, se enviará un mensaje de correo electrónico a la dirección especificada. Para ello se han registrado el dominio qplan.it y un servicio de correo electrónico, cuyas credenciales se han especificado en el backend de la aplicación.

Figura 30. Correo electrónico de activación de cuenta

Al pulsar sobre el enlace del mensaje de correo electrónico, se activará la cuenta permitirá iniciar sesión. Lo mismo ocurre al solicitar un cambio de contraseña por olvido, recibiendo un correo con un enlace que nos permite acceder a una página donde poder cambiar nuestra contraseña usando un token temporal de 15 minutos desde el envío del correo. En la figura 31 puede observase la página en cuestión donde se puede apreciar el mensaje que se recibe se accede cuando el enlace ya ha sido utilizado para cambiar la contraseña o ha expirado.

En caso de cambiar la contraseña satisfactoriamente, al disponer el ID de usuario (mediante la URL), se utilizará la contraseña introducida para iniciar sesión, de forma que no será necesario introducir las credenciales nuevamente.

Figura 31. Página para cambio de contraseña

4.5.2. Calendario para selección de días

Uno de los requisitos fundamentales de la aplicación es permitir la mayor flexibilidad posible, por lo que necesitamos un calendario que permita seleccionar un conjunto específico de días, sin embargo, tras investigar todas las soluciones de código libre ya existentes, no hemos encontrado ningún calendario que permita esta característica, ya que todos están destinados a seleccionar un día o, a lo sumo, un rango de días. Implementar un calendario desde cero para conseguir esta funcionalidad requeriría una gran cantidad de tiempo, por lo que finalmente se decidió usar el calendario que más se ajustase a nuestras necesidades como base para modificarlo y construir un componente propio. El calendario que finalmente de escogió fue “react-date-range” (https://github.com/Adphorus/react-date-range), de forma que se permite una selección de un rango de días que posteriormente se transforma en un conjunto de selecciones de un único día que abarque todos los que se han seleccionado, además se permite volver a seleccionar un día para eliminarlo y limpiar todas las selecciones mediante un botón en la parte inferior. En la figura 32 se muestra el resultado final.

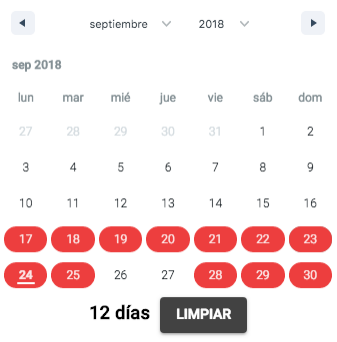


Figura 32. Calendario personalizado

Por último, dado que la funcionalidad anteriormente comentada se encarga de proporcionarla nuestro componente personalizado, nos permite reutilizarlo en distintas partes de la aplicación de una forma muy sencilla, tal y como se puede observar en la figura 33.

```
<Calendar
  selectedDates={props.selectedDates}
  onSelectedDatesUpdated={props.onSelectedDatesUpdated.bind(this)}
  appendDates={true}
  selectRange={true}
  primaryColor={theme.palette.primary.main}
  secondaryColor={theme.palette.secondary.light}
/>
```

Figura 33. Componente del calendario

4.5.3. Selector de rangos de tiempo

Se trata del componente más complejo que se ha desarrollado para la aplicación, éste permite seleccionar los tramos horarios en los que se podrá llevar a cabo del evento, indicando una hora de inicio, final y permitiendo bloquear ciertos tramos del día.

El objetivo de ese componente es permitir la mayor flexibilidad posible sin sacrificar la facilidad de uso, por lo que por defecto únicamente muestra una hora de inicio y final del evento, ocultando la selección de tramos bloqueados. No obstante, podemos desplegarla haciendo clic sobre la caja de selección correspondiente. En la figura 34 se muestra los dos estados principales del componente.

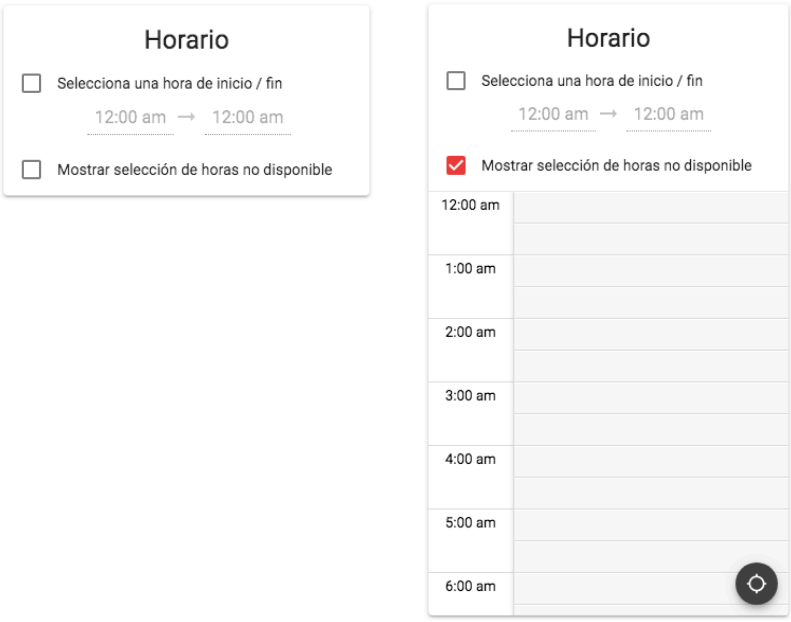


Figura 34. Selector de rangos de tiempo

Este componente permite seleccionar los tramos bloqueados dentro de las horas indicadas simplemente arrastrando con el ratón (o mediante la pantalla táctil de un dispositivo móvil) las horas que deseamos bloquear. Además, permite reflejar un evento que acabe el día siguiente sin necesidad de dividir el evento en dos. Sin embargo, dado que se ha usado como base el calendario de código libre "react-big-calendar" (https://github.com/intljusticemission/react-big-calendar), no se permite mostrar horas del día siguiente en el mismo grid, por lo que el componente realiza la transformación necesaria mediante el cálculo de un desplazamiento (offset) para trabajar con un grid que comience a las 00:00 horas, pero al usuario se le muestra la hora que corresponda con la seleccionada. En el momento de la selección de un tramo horario, se realiza el proceso inverso, sumando el desplazamiento y devolviendo la hora que realmente se había seleccionado.

De esta forma se consigue la sensación de que un evento que comienza un día y acaba el siguiente, por ejemplo, comenzando a las 23:00 y acabando a las 02:00 del día siguiente, realmente se trata de un único evento, sin necesidad de separarlo en dos días distintos. Además, se permite una selección más precisa, pudiendo bloquear tramos con una precisión de hasta 5 minutos. Todo esto puede observarse en la figura 35.

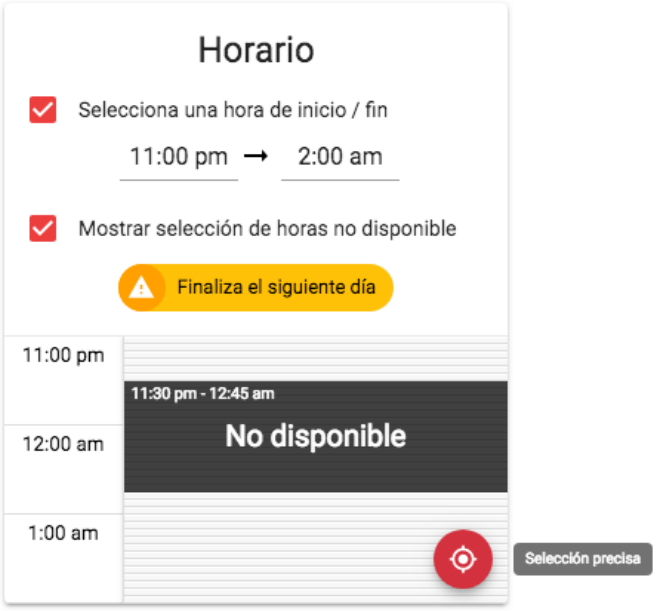


Figura 35. Selección precisa y alertas del selector

A modo de advertencia se indica que el evento finalizará el día siguiente, tal y como se puede ver en la figura anterior, además de, en el caso de que las horas del día siguiente se solapen con las del actual (como ocurre en el caso de la figura 36), no sólo se mostrará una alerta indicando que comenzará cuando finalice el día anterior, sino que podemos ver los rangos bloqueados de tal día (estas selecciones aparecerán con una alerta en la esquina superior derecha indicando que pertenece al día anterior). Además, no podremos modificar ese tramo horario, sino que debemos hacerlo desde el día correspondiente.

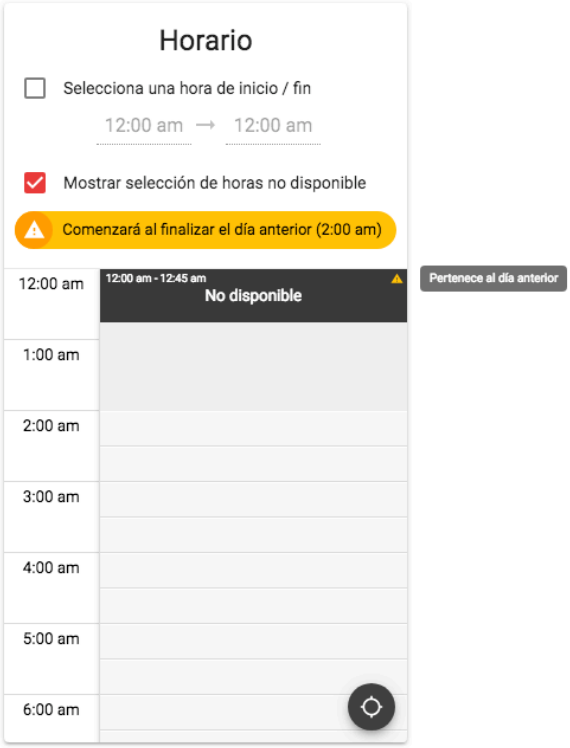


Figura 36. Eventos anteriores y alertas del selector

Nos gustaría indicar que en el caso de que se haya seleccionado unos tramos bloqueados en el día siguiente y se modifique la hora actual para que se produzca un solapamiento, se recalcularán los tramos bloqueados para que comience y termine en la fecha correcta. Además, podemos anular un día por completo, de forma que ese día se eliminará en el momento de enviarse al servidor sin necesidad de volver al paso anterior y deseleccionarlo, tal y como se muestra en la figura 37.

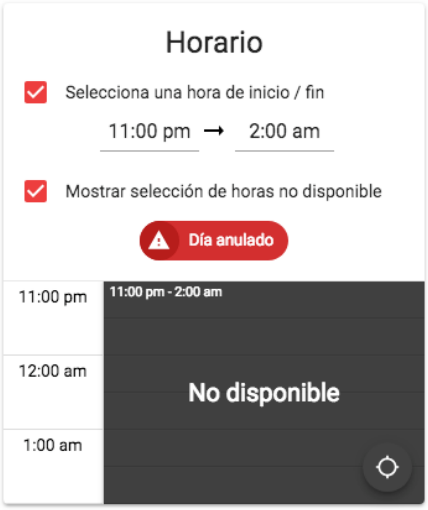


Figura 37. Día anulado en el selector

Como se habrá podido intuir, también podemos bloquear ciertos tramos horarios por cada uno de los días seleccionados, permitiendo una gran flexibilidad. Para ello tan sólo se deberá seleccionar el día deseado mediante el calendario (reutilizado de la característica anterior) situado a la derecha.

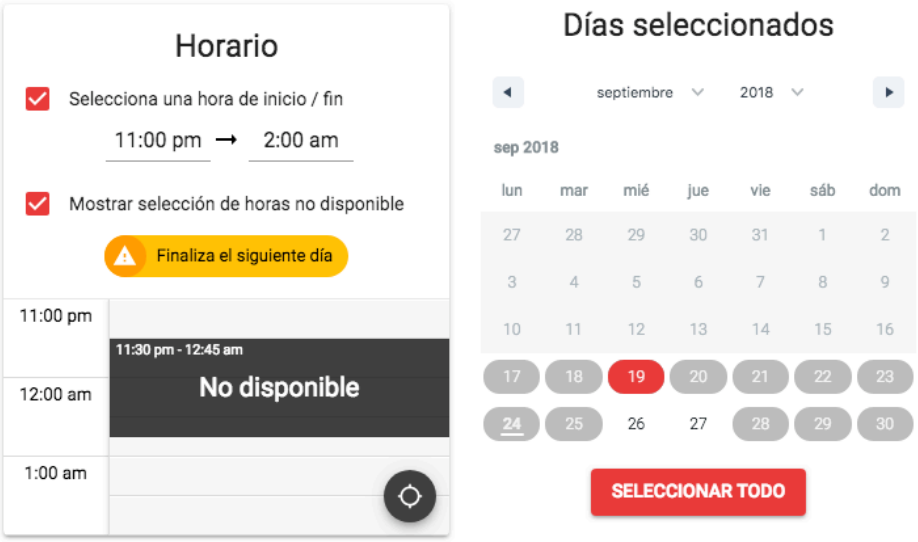


Figura 38. Selector de rangos de tiempo y calendario

En el caso de hacer clic en el botón de “Seleccionar todo” se comprobará si todos los días disponen de la misma configuración horaria y en caso negativo, se preguntará por reiniciarlas y permitir modificar la configuración de todos los días al mismo tiempo.

Estas son las principales características del componente, sin embargo, se tenido en cuenta todas las posibilidades a la hora de seleccionar los tramos horarios para que el componente funcione de la forma que se espera, consiguiendo que sea lo suficientemente simple como para crear un evento rápido y lo suficientemente detallado como para especificar un evento con unas planificaciones horarias muy concretas.

4.5.4. Proceso de creación del evento

Tal y como se ha comentado en distintas ocasiones en este documento, se ha impuesto como requisito la sencillez de uso, por lo que se ha diseñado la creación del evento con el menos número de pasos posible, consistiendo tan sólo en tres pasos, de los cuales uno de ellos es opcional.

Antes de comenzar se necesita especificar un título para el evento que se desea realizar, esto se podrá hacer un cuadro de texto disponible en la página principal para que tras pulsar el botón “Crear evento”, podamos comenzar con la creación.

Figura 39. Página principal de la aplicación

También nos gustaría indicar que, en el caso de dejar la creación de un evento a medias, queda guardada la información, apareciendo el título que pusimos en un principio y cambiado el texto del botón a “Continuar evento”.

A continuación, podremos seleccionar los días que permitiremos seleccionar a los participantes mediante el calendario que detallamos anteriormente.

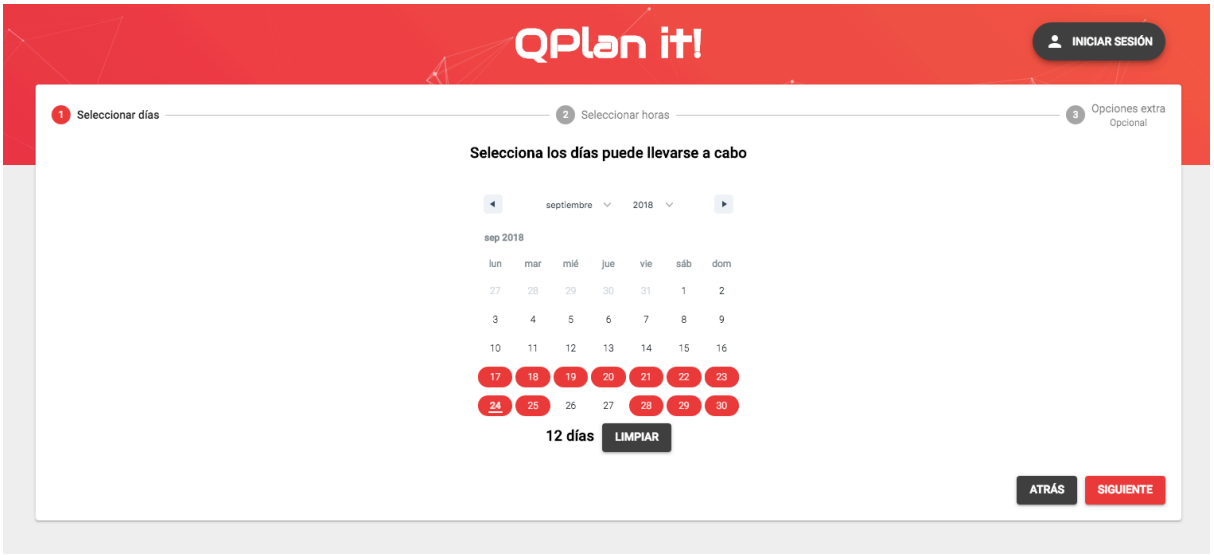


Figura 40. Selección de días para el evento

Una vez seleccionados los días disponibles, podremos establecer la configuración horaria, de forma que podemos avanzar simplemente y permitir seleccionar cualquier hora de los días indicados o seleccionar una configuración específica por cada día.

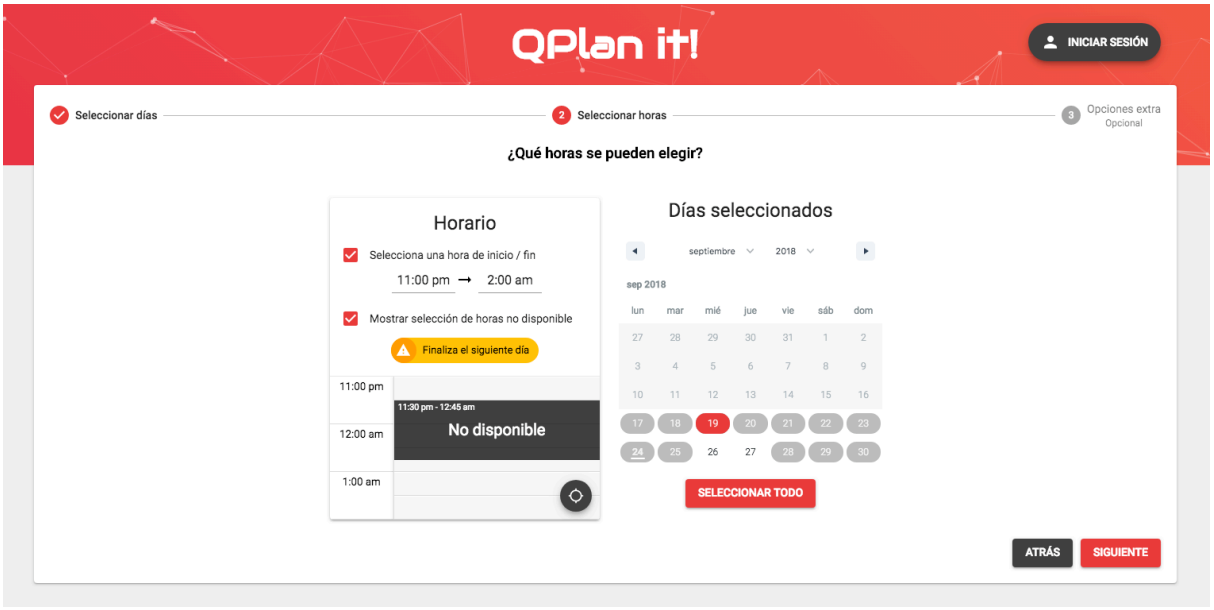


Figura 41. Selección de horas para el evento

Llegados a este punto, si queremos volver atrás para deseleccionar un día, podremos hacerlo, pero aparecerá un mensaje indicando que se perderá la configuración horaria y, por lo tanto, pidiendo una confirmación. De esta forma evitamos perder información de forma involuntaria.

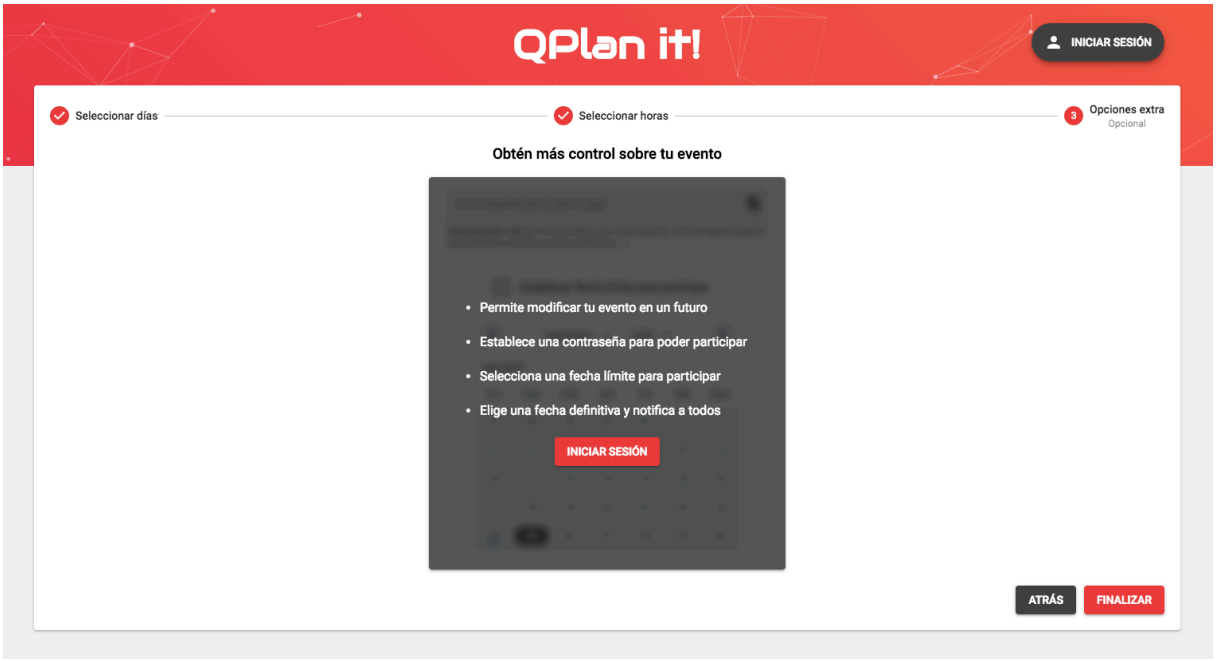


Figura 42. Opciones adicionales (sesión sin iniciar)

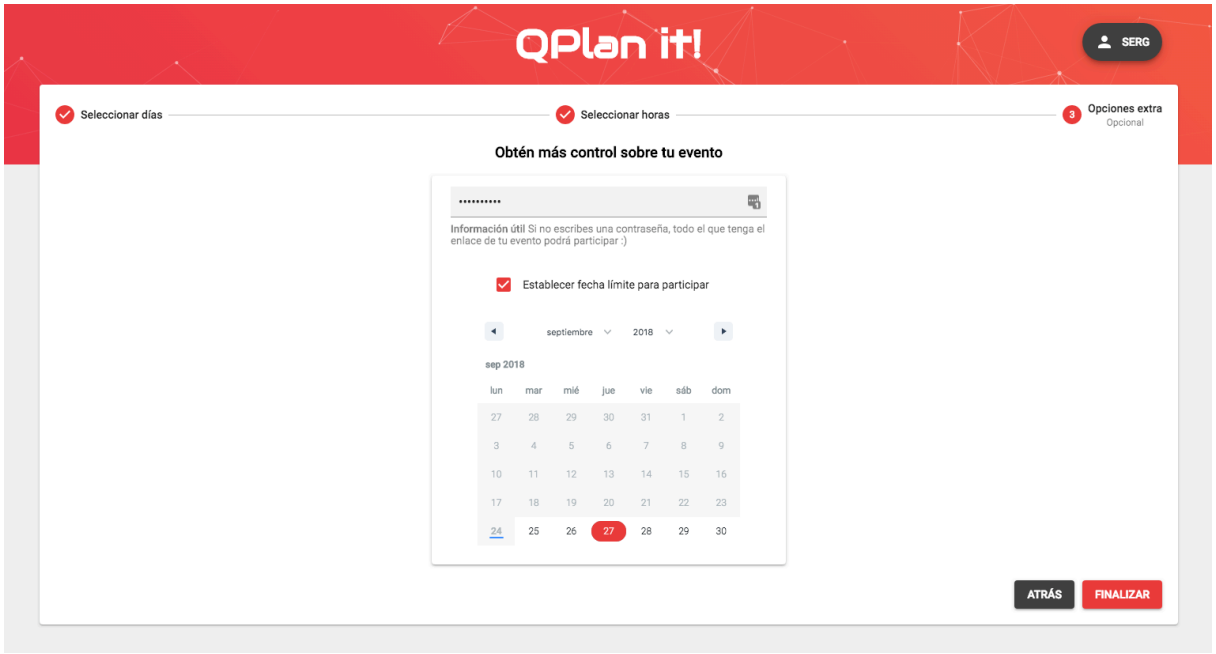


Figura 43. Opciones adicionales (sesión iniciada)

Finalmente llegamos al tercer paso en el que podremos establecer una contraseña y seleccionar la fecha límite de participación. Estas características son opcionales y se necesitará iniciar sesión para desbloquearlas, permitiendo además asignar el evento a nuestro usuario, y así realizar futuras modificaciones, además de elegir una fecha definitiva para la realización del mismo. En las figuras 42 y 43 puede observarse el contenido de este paso cuando no se ha iniciado sesión y cuando no se ha hecho.

Al hacer clic en finalizar, se realizará una llamada al servicio oculto de re-captcha que validará el que ha originado la petición se trata de una persona, en cuyo caso nos proporcionará un token que enviaremos al backend de la aplicación. Éste comprobará su validez y creará el evento, permitiendo realizar una redirección hacia la página del evento.

4.5.5. Participaciones de un evento

Antes de entrar en detalles sobre cómo participar en un evento, nos gustaría comentar que, tal y como se especificó en el backend de la aplicación, en el caso de crear un evento sin haber iniciado sesión, es posible reclamar el evento como de un determinado usuario, empleando un token de reclamación que se nos proporciona en el momento de su creación. Esto se muestra mediante un botón que nos permite asignar el evento a nuestro usuario.

Figura 44. Botón para reclamación de evento

Al pulsarlo nos aparece una ventana emergente que nos solicita de una forma amigable que iniciemos sesión para posteriormente poder intercambiar el token de reclamación por la asignación del evento, tal y como puede observarse en la figura 45.

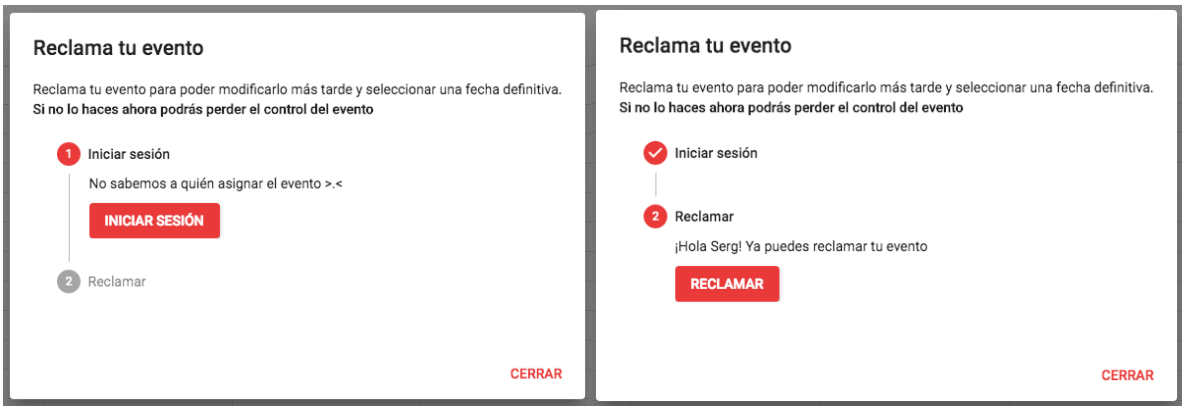


Figura 45. Ventana emergente para para reclamación de evento

Tras la reclamación, desaparecerá el botón y aparecerá el mensaje “Eres el propietario”.

En cuanto al evento, es posible participar de forma anónima, para lo cual se solicitará al menos el nombre del participante mediante una ventana emergente. Una vez indicado el nombre, aparecerá bajo el título del evento, el mensaje “Participando anónimamente como: {Nombre}” y un botón para modificarlo. Si por el contrario hemos iniciado sesión, no aparece ningún mensaje.

Figura 46. Ventana emergente de participación anónima

En el evento se muestra un calendario formado un por el mismo componente que se diseñó para el selector de rangos de tiempo y que hemos detallado anteriormente. En este caso se muestra un máximo de 7 componentes, permitiendo la navegación entre los días disponibles mediante unas flechas en la parte superior.

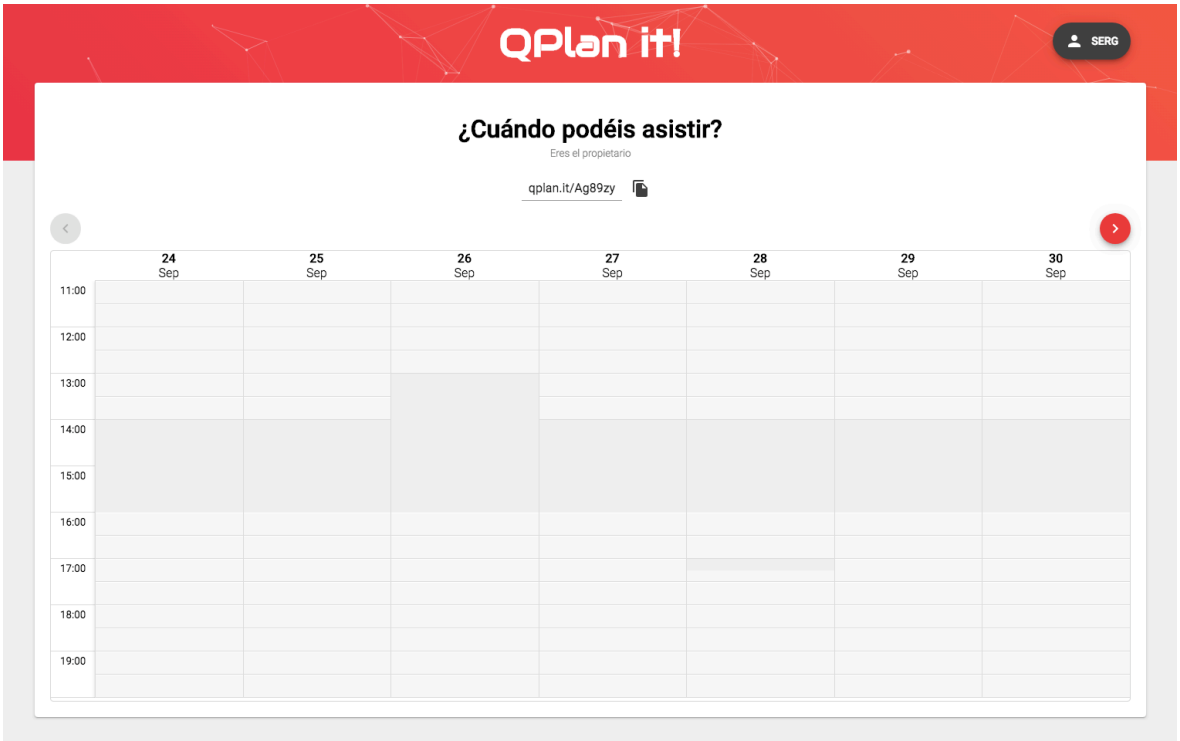


Figura 47. Calendario de evento sin selección

Como se puede observar, los tramos no disponibles aparecen en un color más oscuro, no permitiendo realizar una selección en dichos tramos. Además, nos gustaría especificar ciertos detalles.

- En el caso de que un día comience y/o termine a una hora distinta que los demás, también se mostrará como un tramo bloqueado que no permita su selección
- Si un día acaba el día siguiente y éste no pertenece a los días seleccionados, se mostrará igualmente en el calendario con el propósito de permitir seleccionarlo.
- Tanto los tramos bloqueados como las selecciones realizadas por los participantes se adaptarán a la zona horaria correspondiente, realizándose las comunicaciones con el servidor con fechas en formato UTC.

Por último, mostramos una figura en la que se muestra la selección realizada por el usuario y, en el caso de que otros usuarios seleccionen su disponibilidad, se mostrará el número de personas que están disponibles en ese tramo horario, además de usarse un código de colores para detectar visualmente la mejor fecha para la realización del evento (rojo: poca disponibilidad, naranja: disponibilidad media, verde: alta disponibilidad).

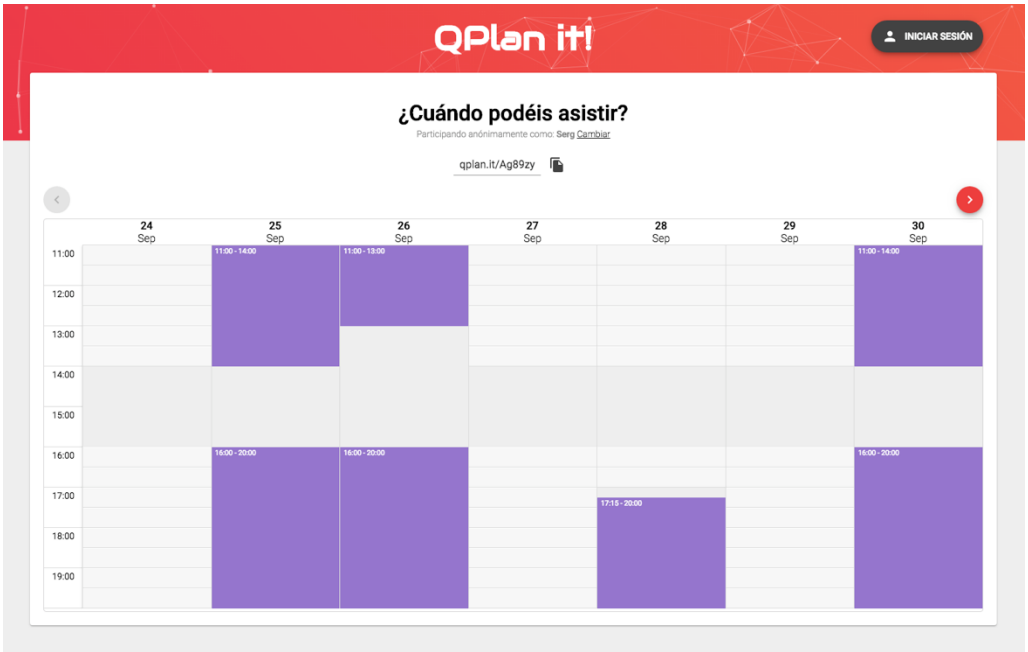


Figura 48. Calendario de evento con selección

5. Conclusiones y líneas futuras

Durante el desarrollo del proyecto nos hemos encontrado con diferentes retos que no han sido fáciles de solventar: desde el planteamiento mediante mockups del sistema que se utilizará para permitir la participación en un evento con la mayor flexibilidad y facilidad posible, hasta cómo se debe operar con fechas de forma que la interfaz de la API REST proporcionada por el backend sea sencilla y funcione como se espera, independientemente de la zona horaria desde la cual se realice la petición.

Aun sí, consideramos que hemos conseguido diseñar e implementar de forma satisfactoria tanto un backend como frontend web para la planificación horaria de eventos que resuelva las carencias de las herramientas disponibles, permitiendo una mayor flexibilidad sin renunciar a la facilidad de uso y escalabilidad.

Para finalizar, nos gustaría comentar que este proyecto nos ha resultado muy enriquecedor, permitiendo usar tecnologías que no habíamos utilizado hasta el momento, solventar retos que no nos había surgido antes e incluso participar en el desarrollo de un componente React de código libre en GitHub, realizando la contribución que se puede encontrar en el siguiente enlace.

https://github.com/TeamWertarbyte/material-ui-time-picker/pull/18

Además, nos gustaría indicar que se pretende publicar la aplicación usando el dominio qplan.it, de forma que cualquier usuario se pueda beneficiar del estudio y desarrollo de este proyecto.

De igual forma, consideramos que esta herramienta se puede mejorar en distintos aspectos, tanto por parte del backend como frontend, por lo que enumeraremos una serie de aspectos que se llevarían a cabo en caso de disponer de una mayor cantidad de tiempo y/o recursos.

Inicio de sesión mediante OAuth2

La aplicación dispone de un sistema de gestión de usuarios independiente, permitiendo el registro y activación de nuevos usuarios, aun así, se podría implementar de una forma sencilla un inicio de sesión mediante OAuth2 usando los

servicios proporcionados por Google y Facebook entre otros, no siendo necesario el envío de un mensaje de correo electrónico para activar la cuenta.

Integración con Google Calendar
Se puede realizar una integración con Google Calendar, siendo una función reservada exclusivamente para los usuarios registrados, no sólo para incentivar el registro sino para guardar el token de acceso al calendario de forma permanente.

Bot para la aplicación Telegram
Dado que disponemos de un backend independiente al cliente, que ofrece una API REST de acceso público, se podría implementar un bot para la aplicación Telegram que facilitase la creación de un evento y donde todos los participantes de un grupo pudiesen seleccionar la opción más adecuada para ellos, esto es posible gracias a que no es necesario un registro previo. Además, en caso que un usuario necesite realizar una selección más precisa, permitiría abrir el evento completo usando el cliente web o nativo de la plataforma.

Aplicación nativa de Android y iOS
El cliente de la aplicación se ha desarrollado usando el framework React y se ha se ha cumplido con las especificaciones del diseño Responsive Design, lo que permite una correcta visualización en dispositivos móviles, de forma que, tras generar una versión de producción, obtenemos un único archivo que permite ejecutar el frontend por completo.

Esto nos permite usar herramientas como Cordova, que nos da la posibilidad de construir una aplicación nativa para los sistemas operativos Android e iOS enmarcando nuestro frontend en un navegador web a pantalla completa, proporcionando así la sensación de ser una aplicación nativa sin necesidad de disponer de varios proyectos independientes.

Mejoras en la API REST
Actualmente se permite obtener todas las participaciones de un evento, en las cuales se incluyen las selecciones que se han realizado. Esto permite que el cliente filtre y agrupe aquellas selecciones que se desea mostrar, pero esto provoca que el servidor envíe demasiada información en aquellos eventos que contengan un gran número de participaciones, información que no siempre es relevante, por lo que se podría

proporcionar un servicio REST que permitiese obtener las participaciones, ya agrupadas, de únicamente aquellos días que se deseen mostrar.

Esta característica provocaría que el servidor tuviese que realizar cálculos adicionales, aunque por otra parte se enviará una cantidad considerable menor de información, además de reducir en gran medida las operaciones que necesita realizar el cliente, por lo que la mejora para el usuario sería sustancial.

Mejoras en el frontend

El frontend de la aplicación permite realizar distintas mejoras que mejoraría la experticia del usuario, por lo que se enumerará alguna de ellas a continuación.

- Incluir una caja de selección, o checkbox, que permitiese la selección de un día completo, con un solo clic o pulsación.
- Incluir un calendario con vista mensual en los eventos para facilitar la navegación en aquellos eventos que dispongan de una gran cantidad de días.
- Incluir un panel de usuario en el que se muestre los eventos que ha creado o participado.

Bibliografía

Exemplary real world application built with React + Redux
https://github.com/gothinkster/react-redux-realworld-example-app

Getting started with create-react-app, Redux, React Router & Redux Thunk
https://medium.com/@notrab/getting-started-with-create-react-app-redux-react-router-redux-thunk-d6a19259f71f

Documentación del paquete "react-localize-redux"
https://ryandrewjohnson.github.io/react-localize-redux-docs/

Material-UI. React components that implement Google's Material Design.
https://material-ui.com/

10 Tips for Better Redux Architecture
https://medium.com/javascript-scene/10-tips-for-better-redux-architecture-69250425af44

A "Composable" React/Redux File Structure?
https://medium.com/@yazeedb/a-composable-react-redux-file-structure-576dcdcfbcd9

A Soundcloud client built with React / Redux
https://github.com/andrewngu/sound-redux/

Large open source react/redux projects?
https://www.reddit.com/r/reactjs/comments/496db2/large_open_source_reactredux_projects/

Scaling your Redux App with ducks
https://medium.freecodecamp.org/scaling-your-redux-app-with-ducks-6115955638be

A react+redux example project based on the re-duck folder structure
https://github.com/FortechRomania/react-redux-complete-example/

Determine a User's Timezone
https://stackoverflow.com/questions/13/determine-a-users-timezone

List of ISO 639-1 codes
https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes